UNIVERSIDAD DE ALCALÁ
DEPARTAMENTO DE AUTOMÁTICA

# RELIABILITY OF PERFORMANCE MEASURES IN TREE-BASED GENETIC PROGRAMMING: A STUDY ON KOZA'S COMPUTATIONAL EFFORT

*Dissertation written by*
David Fernández Barrero

*Under the supervision of*
Dr. María Dolores Rodríguez Moreno
Dr. David Camacho Fernández

Dissertation submitted to the School of Computing of
the University of Alcalá, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
November
2011

*Para mis padres, Manuel y Milagros,*
*por su amor incondicional*
*e infinita comprensión*

*y para Ignacio Criado,*
*por enseñarme que la ciencia*
*nace de la pasión*
*por el conocimiento*

*To kill an error is as good a service as,*
*and sometimes even better than,*
*the establishing of a new truth or fact.*

Charles Darwin

# ABSTRACT OF THE DISSERTATION

Reliability of performance measures in tree-based Genetic Programming:
A study on Koza's computational effort

by

David Fernández Barrero

The measure of computational effort was first proposed by John R. Koza in his book *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, as a method to assess algorithm performance. This measure estimates the minimum number of individuals that have to be processed by a generational Evolutionary Algorithm in order to achieve at least one success with a certain given probability. Computational effort has had a strong influence in the Genetic Programming community, and has been widely used as a performance measure.

Several researchers have shown some concerns, through informal channels, about the behaviour of this measure, but there is little evidence in the literature to support this perception. *This PhD thesis is an attempt to determine whether the concerns about the reliability of Koza's computational effort are solid* and therefore it is a unreliable measure, or, on the contrary, these concerns have no sound support. In order to answer whether computational effort is reliable or not, the goal of the dissertation is to model the error associated to the estimation of this measure. The developed model is essentially theorerical, but some parts are based on empirical evidence.

The main conclusion of the thesis is that *there are sound reasons to doubt the reliability of Koza's computational effort* and should therefore no longer be used. Other simpler measures, such as the success probability or the average number of evaluations to a solution should be used instead.

Futher contributions include the proof that the success rate in Evolutionary Computation is binomially distributed; a characterization of some binomial confidence interval methods; a new method to estimate the success probability; and a run-time analysis of tree-based Genetic Programming. In addition, some methods to solve real world problems in the domains of language induction, RFID and logistics are developed.

# RESUMEN AMPLIADO DE LA TESIS

Reliability of performance measures in tree-based Genetic Programming:
A study on Koza's computational effort

por

David Fernández Barrero

El esfuerzo computational de Koza es una medida de rendimiento algorítmico ampliamente utilizada en la Programación Genética (PG). Dicha medida estima el número mínimo de evaluaciones que son necesarios para que un algoritmo encuentre al menos una solución con una cierta probabilidad. Esta medida, por diversos motivos, ha ejercido una notable influencia en el desarrollo de la PG como disciplina. Sin embargo, existe una considerable discrepancia entre la importancia del esfuerzo computacional y el conocimiento disponible sobre sus propiedades.

A través de canales informales, diversos investigadores han mostrado reticencias debido a ciertas anomalías observadas en el comportamiento del esfuerzo computacional, aunque dicha preocupación no está respaldada por evidencia empírica o teórica. Esta tesis en un intento de aumentar el conocimiento sobre dicha media. Más concretamente, *se plantea como pregunta de investigación principal determinar hasta qué punto el esfuerzo computacional es una medida de rendimiento fiable*. Con el fin de poder perfilar una respuesta fundada, se plantea como objetivo de la tesis obtener una caracterización del error asociado a la estimación del esfuerzo computacional.

Se identifican dos fuentes de incertidumbre en la estimación del esfuerzo computacional: el operador de redondeo, y el error de estimación. Se demuestra analíticamente, con respaldo empírico, que el *operador de redondeo* introduce un error absoluto máximo igual al producto de la generación y el tamaño de la población. En cambio, en términos relativos el error de redondeo está acotado por una función no lineal monótona creciente con la probabilidad de éxito del algoritmo. El error inducido por el operador de redondeo tiene una forma trivial de eliminarse consistente en no utilizarlo.

El *error de estimación* es la única fuente de aletoriedad en el proceso de medida, y es intrínseco al mismo. Su origen se sitúa en la estimación de la probabilidad de éxito, de la que depende el esfuerzo computacional. Caracterizar el efecto de dicha estimación en el esfuerzo computacional no es trivial, y require un modelo analítico de la probabilidad de éxito. El modelo propuesto en la tesis se basa en la descomposición de la probabilidad de éxito en dos

términos, que a su vez modelan aspectos distintos de la probabilidad de éxito. Un primer término modela la probabilidad de que el algoritmo obtenga un éxito al final de su ejecución. Dicho término no depende del tiempo y por lo tanto lo denominamos estático. El segundo término modela la evolución de la probabilidad de éxito en el tiempo, y por lo tanto es dinámico. Deducir las propiedades estadísticas del modelo propuesto tiene dos dificultades, la primera es que se necesita caracterizar el error de estimación de los parámetros de una distribución binomial, y por otra parte obtener una caracterización estadística del tiempo que un algoritmo tarda en encontrar una solución.

El *término estático tiene una naturaleza binomial*, y por lo tanto la estadística binomial puede aplicarse. Siendo más precisos, nos intersa caracterizar la incertidumbre asociada a la estimación de la probabilidad, y una forma de hacerlo es por medio de intervalos de confianza binomiales. Como resultado de estudiar cuatro métodos de cálculo de intervalos binomiales (aproximación a normal, Agresti-Coull, Wilson y "exacto"), se comprueba que el método de Wilson presenta un buen comportamiento medio, y por lo tanto es una opción razonable para caracterizar el error de estimación de la probabilidad estática de éxito. Como aplicación directa del resultado, se obtiene una estimación de la calidad de la medición de la probabilidad en función de la probabilidad estimada y el número de ejecuciones.

El *término dinámico* necesario para modelar la probabilidad de éxito depende del tiempo que tarda un algoritmo en encontrar una solución, que es desconocido. Afortunadamente es un problema fácil de solucionar utilizando una aproximación experimental. Utilizando una serie de problemas clásicos en PG, se determina que existen tres distribuciones estadísticas que modelan adecuadamente el comportamiento dinámico de la probabilidad de éxito. La distribución que aparece en un mayor número de casos analizados, incluyendo aquellos que podemos considerar más comunes, es la distribución lognormal. En ciertos casos extremos, también aparecen las distribuciones exponencial y Weibull. La primera aparece en problemas booleanos difíciles, si no se consideran las ejecuciones que encuentran una solución durante la fase inicial. Por el contrario, la distribución de Weibull aparece asocida al tiempo que los casos analizados tardan en encontrar la solución en ausencia de presión selectiva.

En base a estas observaciones, *se propone un nuevo método para modelar y estimar la probabilidad de éxito de un algoritmo*. Si bien las pruebas experimentales no aportan evidencia de que el nuevo método mejore la estimación clásica de máxima-verosimilitud, al menos sí la iguala en cuanto a la exactitud. Adicionalmente es capaz de interpolar y extrapolar valores de probabilidad, lo que tiene como resultado una función de probabilidad menos abrupta, especialmente cuando el número de éxitos disponibles para calcular la estimación es reducido. Por lo tanto, dicho modelo presenta unas propiedades razonables para utilizarlo en la caracterización del error de estimación del esfuerzo computacional.

En base al modelo elaborado, se determina analíticamente que la precisión del esfuerzo computacional es poco sensible al tiempo de ejecución del algoritmo, pero sí aparece una dependencia significativa con la varianza del mismo, tanto mayor cuanto mayor es la varianza. En todo caso, tanto el modelo analítico, como los resultados experimentales, muestran que con el número de ejecuciones habitualmente utilizadas, en torno a $50$, *el error de estimación del esfuerzo computacional suele ser apreciable*.

Complementariamente al objeto principal de estudio de esta tesis, de carácter básico, se realiza una investigación aplicada. En particular, se parte de una plataforma de extracción e integración de información basada en agentes semánticos llamada Searchy, para exten-

derla incorporándole la capacidad de evolucionar expresiones regulares. Como base para seleccionar el alfabeto del que se nutre un algoritmo genético, se propone un nuevo algoritmo inspirado en la ley de Zipf. Por último, se han aplicado técnicas evolutivas en planificación logística y RFID.

Las operaciones no lineales a las que se somete a la probabilidad de éxito en el cálculo del esfuerzo computacional induce comportamientos asintóticos. Por lo tanto se concluye que, en determinadas circunstancias, *pequeños errores de estimación de la probabilidad de éxito, se traducen en errores considerables de la estimación del esfuerzo computacional*. Si se considera el escaso valor añadido que aporta el esfuerzo computacional en relación a otras medidas de naturaleza más básica, que carecen de estos defectos, concluímos que *la utilización del esfuerzo computacional debería de evitarse en la medida de lo posible*. En caso de que el uso del esfuerzo computacional sea necesario, se aconseja eliminar el operador de redondeo, y ajustar el número de ejecuciones en función del error admisible en la experimentación.

# Agradecimientos
## Acknowledgements

A lo largo de mi vida he afrontado proyectos pequeños y grandes, ambiciosos y no tanto, alguno loco y bastantes ingenuos, sin embargo, ninguno de ellos ha sido tan radicalmente individual como el doctorado. Realizar una tesis doctoral supone un ejercicio de intromisión singular, propio de la intensidad intelectual que se supone en un doctorado. Sin embargo, he aquí una paradoja: a pesar de ese ejercicio de radical individualidad, una tesis es a su vez una obra colectiva. Esto es especialmente cierto en esta tesis, puesto que sin la aportación de tantas personas, en distinto grado e implicación, nunca habría podido sustanciarse. De justicia es reconocer, y agradecer, estas aportaciones.

En primer lugar quisiera mencionar a quienes me han rodeado profesional y académicamente en el Departamento de Automática de la Universidad de Alcalá. Mi *alma mater*, un lugar en el que me formé gracias a la labor de unos no siempre suficientemente reconocidos docentes, algunos de los cuales ahora tengo el placer de tener como compañeros y amigos. Quisiera recordar especialmente a aquellos compañeros que entienden que la Universidad ejerce una labor fundamental en la generación y transmisión del conocimiento, y que, por lo tanto, se juegan unos siempre difíciles equilibros entre la investigación y la docencia.

Siguiendo en el ámbito académico, quisiera reconocer a los integrantes de los grupos de investigación GSIC de la Universidad de Alcalá: Yolanda, Daniel, Pablo o Miguel, entre otros; y AIDA, de la Universidad Autónoma de Madrid: Raúl, Héctor, Antonio o Gema. Tengo el íntimo y sólido convencimiento de que en el seno de ambos grupos se está formando a una brillante generación de investigadores. Quisiera agradecer en particular a Bonifacio, por su gran paciencia al atender mis dudas matemáticas, y por compartir conmigo la emoción de encontrar en las Matemáticas la respuesta a tantas preguntas. Sea como sea, espero que en el futuro podamos seguir disfrutando juntos de la investigación, y que mi tesis sea la primera de muchas y mejores tesis doctorales.

Mi familia ha jugado un papel tan clave como particular en la elaboración de la tesis. Su paciencia ante las ausencias, su comprensión, su aliento han sido una pieza clave: Margarita, Concha, Sergio, Mercedes, Braulio, el otro Braulio, Mila, Angelita, Jorge, Félix, Álvaro, Sara, Arturo, Cris o Carlos. Con especial cariño quisiera agradecer el apoyo de mi abuela, Dolores, que a pesar de no saber muy bien qué es eso de una tesis doctoral, la bastó con saber que era algo importante para su nieto, para vivir su desarrollo con la intensidad con la que ha vivido todo este largo proceso doctoral.

Por otra parte, mis amigos, viejos y nuevos. Todos ellos contribuyendo a la realización de la tesis, bien con su apoyo constante, bien con su paciencia por las ausencias. La lista es larga: Rubén y Paco, por todos los paseos veraniegos que hemos dado; Janire, por estar siempre presente en los momentos díficiles, incombustible y paciente; Wendy, por su ilusión, determinación y fortaleza afrontando con valentía una tesis contra toda adversidad, es para

mi un ejemplo a seguir; David y Fu, por los momentos vividos en esa otra aventura que fue convertirse en ingenieros; a Mentxu, por su pasión, su vitalidad, y su admirable capacidad de enredar; Noe, a la que deseo la mayor de las suertes en su ruidosa tesis maternal o Reme, por su persistencia en sus objetivos.

Mención aparte requieren mis amigos nicas, tan lejanos en la distancia como cercanos en emociones. Los momentos vividos en la UNAN-León fueron la motivación de una transformación vital que ha acompañado a mi existencia desde que pisé por primera vez mi amada Nicaragua. Me han enseñado a vivir de otra manera, a vencer miedos, a replantearme verdades que entendía como sólidas, a terminar de convencerme de que la educación puede y debe transformar el mundo en el que vivimos. Ricardo, Raúl, Ernesto, Aldo, Álvaro, Rina, Valeria, Denis, Johanna, Julio, Santiago y tantos otros, todos muy presentes en el día a día de mi vida, y espero que sigan estando por muchos años más.

De mi estancia en Portsmouth quisiera destacar los estimulates retos intelectuales propuestos por Julio, quien además me transmitió la emoción del criptoanálisis, y me descubrió el yasai yaki soba. Tambien quisiera mencionar a Vaughan, por esas conversaciones sobre lo humano y lo divino al regreso de la Universidad, y por haber hecho todo lo posible para que en Portsmouth me sienta como en casa. A la Universidad de Portsmouth tengo que agradecer el haberme cedido espacio y medios para el desarrollo de una parte importante de la tesis.

Es un placer especial reconocer el papel irremplazable de Ignacio, de quien aprendí el concepto de publicación, alguien que es un referente científico, humano y moral. Quisiera agradecerle la aportación decisiva que ha hecho a esta tesis, por aportarme un modelo a seguir, por transmitirme la pasión por la investigación, el amor al conocimiento, y por su apoyo constante. Pero sobre todo, quiero agradecerle dos décadas de una amistad ejemplar, que me ha definido en gran medida como la persona que soy ahora.

Y qué puedo decir de mis padres académicos, Malola y David. Su tutela ha sido, más allá de los tópicos agradecimientos doctorales, modélica. De Malola quisiera destacar su calor humano, su aliento constante, su presencia en todos y cada uno de los muchos momentos amargos, y también en los momentos dulces. A David le estoy agradecido por haberme transmitido la pasión con la que investiga, su gusto por el olor a paper por las mañanas, y la búsqueda incansable de la excelencia. A ambos, les agradezco la oportunidad que me han brindado de ser doctor, porque sin ellos, sin su buen trabajo, sin su calidad humana, sin su esfuezo e incluso sin su ilusión, esta tesis no sería una realidad.

Por último, y muy particularmente quiero agradecer a mis padres todo, porque todo se lo debo a ellos. Su vida ha sido un ejemplo de sacrificio y abnegación por su hijo, para brindarle la mejor educación que pudieron, y que a la postre, ha conducido a esta tesis. Mi padre, Manuel, que desde pequeño intentó inculcarme en la medida en que pudo una visión científica, estimulando mi curiosidad, las ansias por conocer, por encuentrar respuestas, y mucho más importante, por encontrar preguntas. Y mi madre, Milagros, una persona hecha a sí misma, superando innumerables dificultades con inteligencia y amor. He necesitado alcanzar una cierta edad para ser consciente de todo lo que han hecho por mí, y quisiera emplear esta tesis doctoral para decirles algo que por simple, por obvio, por evidente, no digo como debería: Gracias por todo, os quiero.

David Fernández Barrero
Portsmouth, octubre de 2011

# Contents


**1 Introduction** **1**
- 1.1 Darwinian motivation of Evolutionary Computation . . . 1
- 1.2 Motivation of the dissertation . . . 3
- 1.3 Problem statement . . . 3
- 1.4 Research questions . . . 6
- 1.5 Structure of the thesis . . . 6
- 1.6 Publications and contributions . . . 7

**2 EC from an experimental perspective** **11**
- 2.1 Experimental research . . . 12
  - 2.1.1 The role of experimentation in Computer Science . . . 12
  - 2.1.2 Classification of experimental designs . . . 14
  - 2.1.3 A framework to describe experiments . . . 17
- 2.2 The first component of experimental designs: Algorithm . . . 18
  - 2.2.1 Trajectory search algorithms . . . 19
  - 2.2.2 Evolutionary Algorithms . . . 21
  - 2.2.3 Swarm Intelligence . . . 25
- 2.3 The second component of experimental designs: Problem . . . 26
  - 2.3.1 Test suites . . . 27
  - 2.3.2 Instance generators . . . 30
  - 2.3.3 Test suites in Genetic Programming . . . 31
- 2.4 The third component of experimental designs: Parameters . . . 32
  - 2.4.1 Parameter control . . . 33
  - 2.4.2 Parameter tuning . . . 34
- 2.5 The fourth component of experimental designs: Measures . . . 37
  - 2.5.1 Classification of measures . . . 38
  - 2.5.2 Performance measures . . . 39
  - 2.5.3 Genotypic measures . . . 43
  - 2.5.4 Specialized measures . . . 44
- 2.6 Conclusions . . . 45

**3 EC from an applied perspective** **47**
- 3.1 Introduction . . . 48
- 3.2 Searchy architecture . . . 49
- 3.3 Mapping and integration in Searchy . . . 51

3.4 Wrapper based on evolved regular expressions . . . 54
3.4.1 Codification . . . 55
3.4.2 Evolution strategy . . . 55
3.4.3 Fitness . . . 56
3.5 Zipf's law based alphabet construction . . . 56
3.5.1 Preliminary considerations . . . 56
3.5.2 Alphabet construction algorithm . . . 57
3.5.3 Complexity analysis . . . 58
3.6 Evaluation . . . 59
3.6.1 Parameter tuning . . . 59
3.6.2 Regex evolution . . . 59
3.6.3 Data extraction . . . 62
3.7 Other approaches to distributed Information Integration . . . 63
3.8 Conclusions . . . 64

**4 Estimation of the success rate in EC** **67**
4.1 The role of success rate . . . 68
4.2 Issues about the estimation of a probability . . . 68
4.3 Determination of the statistical distribution of SR . . . 69
4.4 Binomial confidence intervals . . . 74
4.4.1 Description of the CIs methods under study . . . 76
4.4.2 Discussion about CI methods . . . 78
4.5 Study on some confidence interval methods performance . . . 78
4.5.1 CIs performance overview . . . 79
4.5.2 Coverage of CIs . . . 80
4.5.3 Average CP . . . 84
4.5.4 Average CIW . . . 84
4.5.5 Discussion of the results . . . 86
4.6 Empirical study on the binomial CIs in tree-based GP . . . 87
4.7 Sample size determination of confidence intervals . . . 87
4.8 Conclusions . . . 90

**5 Run-time analysis of tree-based GP** **93**
5.1 Introduction to run-time analysis . . . 94
5.2 Run-time analysis of tree-based Genetic Programming . . . 95
5.2.1 Run-time behaviour of tree-based GP classical problems . . . 96
5.2.2 Run-time behaviour of tree-based GP with difficult problems . . . 101
5.2.3 Run-time behaviour of tree-based GP with random selection . . . 105
5.3 A simple theoretical model of generation-to-success . . . 107
5.4 A new model of success probability . . . 110
5.4.1 A general model of success probability . . . 111
5.4.2 A specific model of success probability . . . 112
5.4.3 Experimental validation of the model . . . 113
5.5 Run-time analysis in other Metaheuristics . . . 115
5.6 Discussion . . . 117

5.7 Conclusions . . . . . 118

**6 Accuracy of Koza's performance measure** **121**
6.1 Introduction . . . . . 122
6.2 Koza's performance measures . . . . . 123
6.2.1 Discussion about computational effort mathematical properties . . . 124
6.2.2 Constant number of individuals to be processed . . . . . 126
6.3 Exploratory experimental analysis . . . . . 127
6.4 Determination of the variability sources . . . . . 130
6.4.1 Ceiling operator . . . . . 130
6.4.2 Estimation error . . . . . 132
6.5 Characterization of the estimation error of $I(M, i, z)$ . . . . . 135
6.5.1 Relative error induced by the estimation error in $I(M.i, z)$ . . . . . 136
6.5.2 Relative magnitude of $\varepsilon_{est}^{I}$ . . . . . 138
6.6 Characterization of the estimation error of $E$ . . . . . 140
6.6.1 Analytical model of $I(M, i, z)$ and $E$ . . . . . 140
6.6.2 Experimental validation of the analytical model of $E$ . . . . . 140
6.6.3 Using the analytical model of $E$ to characterize its estimation error 143
6.7 Experimental analysis of Koza's performance measures . . . . . 144
6.7.1 Accuracy of $\hat{I}(M, i, z)$ . . . . . 146
6.7.2 Accuracy of $\hat{E}$ . . . . . 150
6.8 Conclusions . . . . . 154

**7 Conclusions and future work** **155**
7.1 Conclusions . . . . . 155
7.2 Future work . . . . . 159

**Bibliography** **161**

# Chapter 1

# Introduction

*Aquí expondré el por qué trato primero de lo primero y segundo de lo segundo*
*y por qué lo tercero ha de ir antes de lo cuarto y después de éste lo quinto*
*Amor y Pedagogía. Miguel de Unamuno*

This chapter motivates and overviews this dissertation. Firstly, we briefly overview Darwin's Evolution Theory as the foundation of Evolutionary Computation. Then, section 1.2 motivates the questions that are addressed later. After that, in section 1.3, Koza's performance measures are briefly introduced in order to provide a basic framework to state the research questions, that are reported in section 1.4. Finally, the main contributions and the associated publications are described.

## 1.1 Darwinian motivation of Evolutionary Computation

The discovery of Evolution Theory is one of the most remarkable achievements of humanity. This theory shaked the dominant position that men had in nature, where they were in a priviliged position in relation to the rest of living beings, to one much more humble, to be just one more species subject to nature's laws. It is difficult to find a scientifical idea able to change the world so deeply as Evolution Theory as stated by Charles Darwin in the *Origin of the Species* [63]. In this book, Darwin made a huge step to increase the knowledge of humanity about humanity. His book, published in 1859, achieved a non comparable success from an editorial an intelectual point of view, being one of the most influential books ever written.

Contrary to what is commonly believed, Darwin was not the first person to postulate that species evolve. The first evolutionary theories date back to some thousands years, in the pre-Socratic ancient Greece. Several centuries after, more elaborated evolution theories emerged: Orthogenesis, Saltationism or Theistic Evolution raised as theories that postulated the existence of forces able to modify the species, or in other words, that species come from

other species. However, all these theories lacked of a scientifical ground. In Darwin's time, evolution was not an strange theory, actually Darwin's grandfather, Erasmus Darwin among other naturalists, envisioned the evolution years before the publication of the *Origin*.

The great contribution of Darwin was not therefore the creation of the concept of evolution, but rather a simpler one: natural selection [65]. He identified natural selection as the driving force in evolution, with other forces such as sexual selection, and took it with all the consequences claiming that human beings were also under the influence of natural selection, as the rest of the species. To some extent, if Copernicus changed the idea of the human being as the center of the Universe, Darwin changed the idea of the human being as the center of nature. Darwinian view of the species represents one of the most notable human achievements in history, one can hardly visualize an intellectual construction with a similar impact in all orders of human existence. In this time of intellectual darkness, it is good to remember the essentials. Darwin gave an extremely simple and elegant explanation that provided an unified view of the position of humans in nature [65, 66], not to mention its central role in several scientifical disciplines, from Geology to Biology, or Psychology.

A surprising discipline where Evolution Theory has been applied with outstanding success is Computer Science. More than one century and a half after the publication of the *Origin*, Darwin's theories motivated a new paradigm in computing. Inspired by Darwin's work, some early computer science researchers, including Alan Turing [231], envisioned an application of his theories to create a new paradigm to solve problems in computing. With sime several algorithms were developed under the inspiration of Evolution Theory, that later became what now we call Evolutionary Computation (EC) [79, 8, 86].

EC belongs to a paradigm in computation that takes nature and natural processes as a source of inspiration. These algorithms are considered generally as general-purpose stochastic search algorithms, and have excellent performance in high dimensionality problems where direct domain-specific algorithms fail [79]. In particular, EC takes the darwinian idea of natural selection as a basis to design algorithms, generally known as Evolutionary Algorithms (EAs). These algorithms, given a set of potential solutions, modify them, select those fittest according to an evaluation function, and use these potential solutions to generate a new population, iterating this process until a feasible solution is found or a budget of resources wasted. Probably, the most popular EA is Genetic Algorithms (GAs) [110, 95]. However, the collection of evolution-inspired algorithms is extense, including Genetic Programming (GP) [136, 192], which plays a central role in this thesis.

GP involves a collection of algorithms whose search is performed in the program search space. More than a theoretically coherent collection of algorithms, GP deals with the problem of program induction [158]. Many different approximations have been used in GP. In particular, the most popular and widely known GP algorithm is the one originally described by John R. Koza in his seminal book [136]. Koza proposed using trees in order to represent programs, without a difference between the phenotypic and genotypic spaces. Due to its simplicity and good results [135], this form of EA has been widely used in practice, and has attracted much research interest, which is translated into a large corpus of literature devoted to this issue, specialized journals, congresses and doctoral dissertations.

## 1.2 Motivation of the dissertation

Despite the algorithmic simplicity of most EAs, their analytical study is extremely difficult [193, 185]. Hence, in order to ease the problem and address a tractable complexity, several assumptions have to be made, limiting the practical suitability of theoretical results [25]. Therefore, research on EC has been heavily supported by experimental approaches, where research is driven by data collection and analysis in controlled environments [59]. In this way, an EA with a certain parameter setting is run to solve a given problem while the experimenter observes the behaviour of the algorithm collecting data for further analysis. Therefore, observation plays a central role, and in a scientifical context, observation means measuring.

In order to understand the behaviour and the performance of EAs, several measures have been proposed and used. These measures capture some characteristic of the algorithm under study, and, depending on the experimenter purposes, measures have to be determined within the experimental design. There is a notable lack of consensus about which measures should be collected [23]. To some extent, it is a logical consequence of the large number of different purposes that the experimenter might have, and the complexity of the EAs.

The lack of consensus when selecting measures has several drawbacks. Comparability of results among experiments reported in the literature is difficult, when not impossible. It also difficulties the understanding and interpretation of the algorithm behaviour through a standard set of measures, that would eventually guide in the algorithm design process. Perhaps more importantly, the lack of "standardized" measures is accompanied by a lack of interest in understanding how the measure itself behaves. Some measurements have side effects whose undertanding is needed in order not to introduce bias in the conclusions. Hooker clearly described this issue, "the problem is one of distinguishing the phenomenon (here, the algorithm) from the apparatus used to investigate it (here, the data structures, code, etc)" [112]. A better knowledge about which tools are used to observe and analyze algorithms behaviour is needed to draw more solid conclusions.

Indeed, the lack of methodological concerns found in the literature is surprising. Many research is devoted to what Hooker named "algorithm race" [112], with quite limited scientifical added value, while these issues, which have a direct impact into research and practice, have attracted little interest [25]; but fortunately there is a change of tendency with an increasing number of publications concerned by methodological and experimental issues [26, 25, 37, 76]. This PhD thesis is an attempt to provide a step forward in this direction, towards a better knowledge about the tools needed in order to improve research in EC. In particular, we address the realiability of a performance measure widely used in GP, *Koza's computational effort*.

## 1.3 Problem statement

One important performance measure widely used by the GP community is the *computational effort*. This measure was originally proposed by John R. Koza in the fourth chapter of his first book [136] among other measures, and was used by him to measure algorithm performance through his books [134, 138, 137]. The impact of this measure has been notable in the GP community. Nonetheless, we can observe that the use of this measure has been decreasing

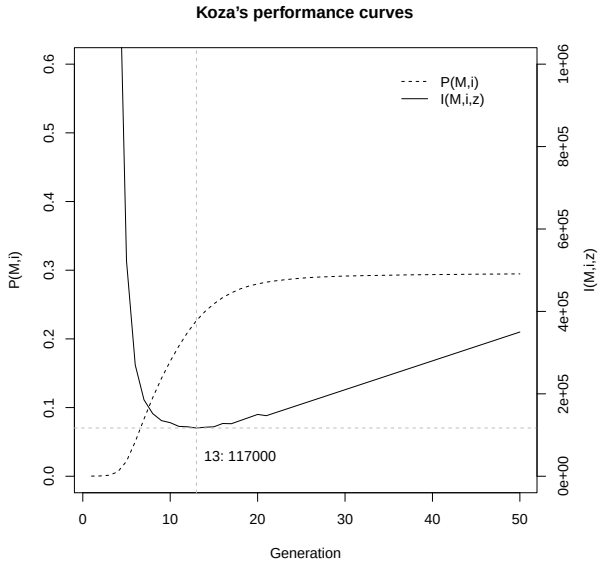


Figure 1.1: Example of Koza's performance curves. Success probability ($P(M,i)$) is represented in dotted line, while $I(M,i,z)$ is the solid line. Computational effort and generation where it is found are also represented.

in the last years. Perhaps, the reason behind this fact can be found in that the researchers are loosing confidence in Koza's measure.

Several researchers have shown their concerns about the reliability of computational effort though informal channels, such as chats in conferences and distribution lists. But surprisingly, these concerns have little support in the literature. *This work is an attempt to check out whether concerns about Koza's computational effort are sound or on the contrary it is a reliable measure*. A detailed description of Koza's computational effort can be found in chapter 6. In order to provide a background needed by the description of the objectives and research questions, we first need to introduce the measure that is the object of our study.

Koza defined *computational effort* ($E$) as the minimum number of individuals that the algorithm has to process to achieve, at least, one success with a given probability $z$. Sometimes this probability is provided using a value $\varepsilon$ such as $z = 1 - \varepsilon$. If the population is composed by $M$ individuals, and the probability of finding a feasible solution at generation $i = 1, ..., G$ is $P(M,i)$, then the computational effort is given by

$$E = \min_i \left\{ Mi \left\lceil \frac{ln(1-z)}{ln(1-P(M,i))} \right\rceil \right\} \tag{1.1}$$

Where $\lceil \ldots \rceil$ stands for the ceiling operator. If the minimum operator of (1.1) is removed, the remaining function is usually denoted by $I(M,i,z)$, so $E = \min_i I(M,i,z)$. It is clear that $E$ and $I(M,i,z)$ are closely related.

Even though computational effort is a scalar value, Koza used a graphical method to report its value. This method, which we name as Koza's performance curves, plots the value

of two related functions that are used to estimate $E$; in particular the accumulated success probability, $P(M,i)$, and $I(M,i,z)$. These two functions are plotted overlapped, and the minimum value of $I(M,i,z)$, which is $E$, is placed there, together with the generation in which this value is found. An example of Koza's performance curves is in Figure 1.1.

A number of non-trivial statistical issues arise when (1.1) is analyzed in more detail. In practice, the value $E$ cannot be known exactly because it has to be measured, and all measure has an associated measurement error. This claim holds specially for EAs, which have an intrinsic stochastic nature.

Measures are random variables, and thus they are subject to variability. In particular, a closer look to (1.1) shows that all the values involved in the computation of $E$ are known, except $P(M,i)$. The ceiling operator removes information, and, depending on the context, might introduce a deterministic bias in the computation of $E$, but it is not random. So, in practice, the definition of $E$ described by equation (1.1) is replaced by

$$\hat{E} = \min_i \left\{ Mi \left\lceil \frac{ln(1-z)}{ln(1-\hat{P}(M,i))} \right\rceil \right\} \tag{1.2}$$

which is what can actually be measured, and in general, $E \neq \hat{E}$.

Given that the only source of randomness of $\hat{E}$ is given by $\hat{P}(M,i)$, the accuracy of $\hat{E}$ depends directly on the accuracy of $\hat{P}(M,i)$. The method to quantify this dependence is well known, by using error propagation. Taking differences in (1.1) w.r.t. $P(M,i)$, we can deduce how an estimation error of $P(M,i)$ would affect $E$,

$$\Delta E = \left| \frac{\partial E}{\partial P} \right| \Delta P \tag{1.3}$$

However, a couple of problems are found when this expression is applied. First, calculating the differential is not trivial given that the analytical form of $P(M,i)$ is unknown. Secondly, it is not clear which $\Delta P$ value should be used, since the estimation error associated to $P(M,i)$ is also unknown. Therefore, in order to understand the performance of computational effort, it seems clear that we have to address the more basic (and general) problem of understanding the statistical properties of estimating success probability.

The characterization of the probability function $\hat{P}(M,i)$ can be decomposed in two different problems. The maximum-likelihood estimator of $\hat{P}(M,i)$ is $k(i)/n$, where $k(i)$ is the number of successful runs in generation $i$, and $n$ is the number of runs executed. Hence, $P(M,i)$ provides information about two facts: How many runs where successfull and when they achieved success. Subsequently, the statistical properties of $P(M,i)$ depend on whether time is considered or not. For convenience, we refer the *static behavior* of $\hat{P}(M,i)$ as the behaviour of the success probability at time $i_0$, i.e., $\hat{P}(M,i_0)$ with $i_0 \in \{1,...,G\}$. On the other hand, the *dynamic behaviour* refers to the run-time behaviour of $\hat{P}(M,i)$, which takes a more complex form since success probability is no longer a random variable, but a discrete-time stochastic process.

The distinction between the static and dynamic behaviour of the success probability is critical in the structure of the thesis. As will be explained in detail in section 1.5, the dissertation core is composed by three chapters: one is dedicated to study the static properties of the estimation of success probability (chapter 4), another studies the dynamic properties of

the success probability (chapter 5) while the third one (6) uses the previous results to characterize the error associated to the measurement of Koza's *computational effort*. With this background, we can state the research questions.

## 1.4 Research questions

The main *research question* that is faced in this PhD thesis can be described as follows.

> *Koza's computational effort has been widely used by the GP community. Despite the concerns shown by several authors about the strange behaviour of this measure, little research has been performed to analyze whether computational effort is reliable or not. The main research question of this dissertation is to determine whether Koza's computational effort is a reliable measure of algorithm performance.*

In order to answer the main research question, we first need to decompose that question into some *specific research questions*:

- **Q1:** Which factors influence the reliability of the computational effort?
- **Q2:** Which statistical properties the static estimation of the success probability has?
- **Q3:** Which statistical properties the dynamic estimation of the success probability has?
- **Q4:** Can the success probability be anatically modeled?
- **Q5:** Does the run-time behaviour provide information about the algorithm?

Based on the previous research questions, we can state the *main goal*:

> *Characterize the estimation error of the computational effort*

Specific research questions are addressed in different chapters, as it is described in the following section.

## 1.5 Structure of the thesis

The dissertation is divided into seven chapters. The first three chapters are introductory. The first chapter is dedicated to introduce the main research question and thesis structure, while the second chapter provides a conceptual framework that helps locating the contributions of this dissertation in the context of EC. Chapter 3 takes an applied perspective and it is dedicated to develop solutions based on EC to some real world problems. This chapter provided the necessary background that motivated the main research question. The chapters that address the research questions are 4, 5 and 6. Following, a more detailed description of the dissertation structure is shown.

- **Chapter 1: Introduction.** It provides a general background, context and motivations. The main objectives and research questions are stated, as well as the dissertation structure, main contributions and publications.

- **Chapter 2: Evolutionary Computation from an experimental perspective.** This chapter is devoted to contextualize the contributions of the PhD thesis. It discusses the role of experimental methods in EC research, and overviews some of the most important issues regarding the experimental design. In particular, this chapter identifies and discusses four components of an experiment: algorithm, parameter setting, problem and measures.

- **Chapter 3: Evolutionary Computation from an applied perspective.** It is an application chapter dedicated to develop methods to evolve regular expressions using GA. The solution proposed uses a semantically driven agent-based information extraction and integration platform named Searchy, which is also introduced.

- **Chapter 4: Estimation of the success rate in Evolutionary Computation.** The statistical properties of the static estimation of the success probability are studied. In particular, the binomial nature of the success rate is investigated. In addition, an extensive study of binomial confidence intervals in the context of EC is provided. This chapter addresses the specific research question Q2. The binomiality of the success rate is used in chapter 5 to characterize the estimation error of the computational effort.

- **Chapter 5: Run-Time analysis of tree-based Genetic Programming.** The statistical properties of the dynamic estimation of the success probability are studied. To be more specific, the time required by a canonical tree-based GP algorithm to find a solution is investigated. As a result of this analysis, an analytical model of the success probability is proposed. This model is used in chapter 5 to provide a close analytical form of the computational effort. Questions Q3, Q4 and Q5 relate to this chapter.

- **Chapter 6: Reliability of Koza's performance measures.** The main research question is addessed providing a characterization of the error associated to the measurement of the computational effort. Additionally, the measurement error of $I(M, i, z)$ is also characterized. This chapter strongly depends on the results obtained in chapter 3 and 4. The specific research question Q1 and the main research question are both addressed in this chapter.

- **Chapter 7: Conclusions.** Research questions are addressed again under the light of the results obtained in the PhD thesis, the conclusions are reported, and some open research lines described.

## 1.6 Publications and contributions

Along the development of this work some publications were generated. In the following, we report them, grouped by the chapter where they appear. In addition, the main contributions are briefly summarized.

- **Chapter 3: Evolutionary Computation from an applied perspective.** This chapter contains some preliminary investigation performed in the context of the dissertation. Despite its objectives are far from the main ones, it served us to acquire experience using EAs, and more importantly, it motivated the main research question the PhD thesis. We have worked in several domains: a logistic application that optimizes the drivers routes [194, 195], generate routes inside a bulding using RFID [196], language induction [98] and data information extraction [49]. For this last application, we have used an agent-based data extraction and integration platform named Searchy [17, 22]. It was used to tune parameters of a GA [13, 17], and simulate a Variable-Length Genetic Algorithm using an island model with immigrants [14]. Finally, evolutive methods to automatically learn regular expresions from a set of positive and negative examples were developed [18, 20].

  - D. F. Barrero, M. D. R-Moreno, and D. Camacho, "Adapting searchy to extract data using evolved wrappers", *Expert Systems with Applications*, To appear, 2011.
  - M. D. R-Moreno, B. Castaño, M. Carbajo, Á. Moreno, D. F. Barrero, and P. Muñoz, "Multi-agent intelligent planning architecture for people location and orientation using RFID", *Cybernetics and Systems*, vol. 42, pp. 16–32, Jan 2011.
  - D. F. Barrero, A. González-Pardo, D. Camacho, and M. D. R-Moreno, "Distributed parameter tuning for genetic algorithms", *Computer Science and Information Systems*, vol. 7, no. 3, pp. 661–677, 2010.
  - D. F. Barrero, M. D. R-Moreno, D. Camacho and B. Castaño "Human drivers knowledge integration in a Logistic Decision Support Tool", in *Proceedings of the 5th International Symposium on Intelligent Distributed Computing (IDC 2011)*, vol. 382/2012 of *Intelligent Distributed Computing*, (Delft, The Netherlands), pp.227-236, Springer-Verlag. 5-7 October 2011.
  - D. F. Barrero, M. D. R-Moreno, and D. R. López, "Information Integration in Searchy: an Ontology and Web Services Approach," *International Journal of Computer Science and Applications (IJCSA)*, vol. 7, no. 2, pp. 14–29, 2010.
  - M. D. R-Moreno, D. Camacho, D. F. Barrero, and M. Gutiérrez, "A Decision Support System for Logistics Operations", in *Soft Computing Models in Industrial and Environmental Applications, 5th International Workshop (SOCO 2010)*, vol. 73, (Guimarães, Portugal, June 2010), pp. 103–110, Springer-Verlag, 2010.
  - D. F. Barrero, A. González, M. D. R-Moreno, and D. Camacho, "Variable Length-Based Genetic Representation to Automatically Evolve Wrappers", in *Proceedings of 8th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2010)*, (Salamanca, Spain), pp. 371–379, Springer-Verlag, April 2010.
  - A. González, D. F. Barrero, M. D. R-Moreno, and D. Camacho, "A case study on grammatical-based representation for regular expression evolution", in *Proceedings of 8th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2010)*, (Salamanca, Spain), pp. 379–386, Springer-Verlag, April 2010.

  - D. F. Barrero, D. Camacho, and M. D. R-Moreno. "A framework for agent-based evaluation of genetic algorithms", in *Proceedings of the 3rd International Symposium on Intelligent Distributed Computing (IDC 2009)*, (Ayia Napa, Cyprus), pp. 31–41, Springer-Verlag. 13-14 October 2009.
  - D. F. Barrero, D. Camacho, and M. D. R-Moreno, *Data Mining and Multiagent Integration*, ch. Automatic Web Data Extraction based on Genetic Algorithms and Regular Expressions, pp. 143–154. University of Technology Sydney, Australia, Springer-Verlag, July 2009.
  - D. Camacho, M. D. R-Moreno, D. F. Barrero, and R. Akerkar, "Semantic wrappers for semi-structured data," *Computing Letters (Cole)*, vol. 4, pp. 21–34, December 2008.

- **Chapter 4: Estimation of the success rate in Evolutionary Computation.** The binomiality of the number of successful runs in EC is proven, with theoretical and empirical support [21, 15]. The usage of confidence intervals to estimate the success rate is discussed [15], and the statistical properties of four binomial confidence interval methods are analyzed in detail [21]. A method to determine the number of runs needed to estimate the success rate with a certain error is proposed [21].

  - D. F. Barrero, M. D. R-Moreno, and D. Camacho. "Statistical Estimation of Success Probability in Evolutionary Computation", *Applied Soft Computing*. To appear. 2011.
  - D. F. Barrero, D. Camacho, and M. D. R-Moreno. "Confidence Intervals of Success Rates in Evolutionary Computation", in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*, (Portland, OR, USA), pp. 975–976, July 2010.

- **Chapter 5: Run-Time Distribution analysis of tree-based Genetic Programming.** The run-time behaviour of tree-based GP is analyced [16], finding that the time required by GP to find a solution is usually follows a lognormal distribution. Based on this observation, an analytical model of success probability is proposed and validated [16]. In order to place this result in the framework of a theory and generalize it, a theoretical model of EA based on Discrete-Time Markov chains is proposed. This model is used to proof that exponentially distributed run-times are a consequence of a memoryless algorithm.

  - D. F. Barrero, B. Castaño, M. D. R-Moreno, and D. Camacho, "Statistical distribution of generation-to-success in GP: Application to model accumulated success probability", In *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, vol. 6621 of *LNCS*, (Turin, Italy), pp. 155–166, Springer-Verlag, April 2011.

- **Chapter 6: Reliability of Koza's performance measures.** The two factors that determine the reliability of the *computational effort* are identified. Based on the results of the chapters $3$ and $4$, the boundaries of the error associated to the measurement of $I(M, i, z)$ and the computational effort are deduced. An experimental validation of these results is also provided in [19]. It is proven that the existence of memory in the algorithm induces a non-constant $I(M, i, z)$.

  – D. F. Barrero, M. R-Moreno, B. Castaño, and D. Camacho, "An empirical study on the accuracy of computational effort in Genetic Programming", in *Proceedings of the 2011 IEEE Congress on Evolutionary Computation*, (New Orleans, LA, USA), pp. 1169–1176, IEEE Press, June 2011.

# Chapter 2

# Evolutionary Computation from an experimental perspective

*For to be possessed of a vigorous mind is not enough; the prime requisite is rightly to apply it. The greatest minds, as they are capable of the highest excellences, are open likewise to the greatest aberrations; and those who travel very slowly may yet make far greater progress, provided they keep always to the straight road, than those who, while they run, forsake it.*
*Discourse on the Method. René Descartes*

This section is devoted to contextualize the contributions of the core chapters of this dissertation, providing a general perspective on the use of the experimentation in the context of EC research. It is not our interest in this chapter to describe a full state-of-the-art on experimental methods, but rather to offer a framework that might help to place the work developed in the core chapters. Indeed, one can hardly talk about a state-of-the-art in this field, as there are some research lines related with some concerns in experimental design, and only a limited number of publications related to the main topic of the dissertation can be found. Nonetheless, the dissertation involves some different research areas. A review of the related literature is reported together, with the contributions.

The chapter begins discussing the role of experimental research in Science in general, and in Computer Science in particular. It motivates the need of using experimentation as a basic tool in research. Then, a classification of the experimental designs in EC is proposed with a four-components framework to describe experiments: Algorithm, problem, parameters, and measures. Then, each one of the components is briefly introduced finishing with measures, the component more closely related to this PhD thesis. Finally, some general conclusions are presented.

## 2.1 Experimental research

Basically, there are two approaches in Science to answer a research question: a theoretical and an experimental approach [25]. These two approaches can be related to the two main philosophical approaches to Epistemology, a topic addressed by philosophers for centuries, from the first Greek philosophers, until Enmanuel Kant and the raise of modern philosophy [123]. Roughly speaking, there were two opposed views. One was initialized by Thales of Miletus and the Ionian school. Thales tried to explain natural phenomena using observation and rejecting any mythological explanation, indeed, it is said to be the first attempt to understand the word with a scientifical basis [209]. Many pre-Socratic philosophers belonged to this school of thought, including Anaximenes, Heraclitus or Anaxagoras.

After Socrates, the position of many philosophers to the problem of how to acquire knowledge changed radically. In particular, Plato's Theory of the Ideas influenced the ongoing philosophy for centuries. Plato saw a natural world as a distorted view of the real word, which is the World of the Ideas, and thus he dismissed observation as a source of knowledge. Following Plato, our senses deceive us from the true. This idea is well represented by the famous Allegory of the Cave [191]. Plato's strong influence in Aristotle and through him to the Schoolmen in the Middle Ages pushed away observation as a source of knowledge. Fortunately, this way of thinking changed in the Renaissance, thus emerging modern Science.

The time dedicated by philosophers to meditate about this topic was necessary to the raise of Science, which has proven to be most effective method to generate knowledge. From a scientifical perspective, observation and abstraction are not two opposing forces, but instead they are complementary. Observation motivates new theories, and new theories motivate new observations. Additionally, observation is used in science as a test of theories, any theory must be consistent with the observations in order to be accepted by the scientific community. In any case, to some extent, the dichotomy between observation and theory remains, and the exact role of each approximation still generates debate. Computer Science is not an exception [70].

### 2.1.1 The role of experimentation in Computer Science

In Computer Science and Artificial Intelligence (AI) researchers may use theoretical or experimental approaches [111, 11]. From an historical perspective, the first research in Computer Science was almost purely theoretical, perhaps because of the academic backgroud of the early AI researchers, most of them coming from Mathematics and Physics, and the hardware limitations of the time. The theoretical origins of Computer Science and AI had a strong influence in these disciplines. In the context of algorithm research, some key authors such as Donald Knuth, in his classical and influential series of books *The Art of Computer Programming*, encourage using analytical analysis of the algorithms [122]. This approach has been so influential that has been the strategy generally followed in algorithm analysis until recently.

Nowdays, the analytical study of algorithms has attracted notable criticism due to its difficulty [175, 25], among other less obvious -and more substantial- drawbacks. Typically, analytical analysis of algorithms use worst-case and average case scenarios [201]. The worst-

case study takes some assumptions that eases the problem, however, by definition the worst-case is a pathological case and therefore it does not represent a realistic scenario [201]. From another perspective, average-case studies are more realistic, but also more difficult, and thus it forces to simplify the problem making it less realistic.

Additionally, these kind of theoretical approaches do not consider details concerning the platform and/or the implementation, which might introduce new elements that potentially can alter the behaviour of the algorithm. This idea was nicely expressed by Hooker, "studying algorithms at a level of a formal system presupposes a form of reductionism, which is the view that one can and should explain a phenomenon by reducing it to its ultimate constituents. Reductionism works in some contexts but fails miserably in others, and I think it often fails in algorithmic science" [111]. Experimentation is a way to overcome those limitations. It is therefore not surprising that algorithmic studies have included experimental methods in their toolbox [59].

Recently, stochastic search algorithms such as Metaheuristics have gained higher popularity. This type of algorithms are particularly difficult to analyze from a theoretical perspective [193, 9, 151]. Despite its algorithmic simplicity, theoretical models and results are scarce and difficult to obtain. The stochastic nature of Metaheuristics introduce a new layer of complexity to analytical studies of this class of algorithms. Actually, there are several authors that complain about the few theoretical works that provide practical results [9], and it is generally assumed that there is no theoretical basis to explain the good performance of Metaheuristics.

Then, due to the challenging complexity and limitations of theoretical approaches, experimental methods have emerged as an alternative to study these algorithms, attracting an increasing research interest. There were several methodological lagoons in early research [31] that motivated to some authors in mid-90s to complain about it, and encouraged using more robust experimental methods. Probably, the most influential work of these early warning papers was written by Hooker [112]. Then, several papers were concerned about the experimental methods [11, 236], tutorial-like papers [92] and even lists of tips to improve experimentation [92, 182].

There are still many methodological concerns about experimentation in EC that motivated more recent publications with similar arguments [76]. In recent years, there has been a considerable effort from several authors to improve experimental methods used in EC research. This increasing interest in experimental methods is materialized in a series of tutorials about experimental methods in the main EC events [248, 249, 31], PhD thesis [206, 36] and monographs [26, 25].

Experiments are undertaken when research questions cannot be answered by direct means [165]. However, it should not be confused with observation. Experiments need observation, but not every observation comes from an experiment. That is the difference between *experimental* and *empirical* research. Data collection makes something empirical [59]. This data is obtained by observing some phenomena, this approach is quite common in some disciplines such as Social Sciences [60, 199]. On the contrary, in *experimental research*, the researcher manipulates the object of study to test it in some desirable conditions, so, the basis of experimentation is not observation, but manipulation [175]. We should mention that this terminology is not generally used, and different terms with similar meanings have been proposed. For instance, Cohen mention observation and manipulation experiments [59] to mean,

respectively, empirical and experimental methods. Similarly, McGeoch [165] distinguishes application and simulation programs. In the following, we will use the term experimental to mean control over the object of study, while empirical will be used as any method that involves observation.

The definition of experiment given by Barr underlines the need of manipulation, "an experiment is a set of tests run under controlled conditions for a specific purpose: to demonstrate a known truth, to check the validity of a hypothesis, or to examine the performance of something new" [11]. But this definition also mentions an important topic: an experiment is just a piece in a more complex machinery that involves research questions, hypothesis, etc. This machinery is named scientific method.

The importance of the method is a general concern in all the scientific disciplines. However, the role of the method in Computer Science in general, and in EC in particular does not seem to generate much debate. Some authors in EC have been concerned about methodological issues, emphasizing, for instance, the sequential nature of experimentation, where an experiment suggests new research questions that, again, suggest new experiments [165]. These methodological considerations are out of the scope of this dissertation, however, some papers about this topic in the context of Metaheuristics and EC can be found in [46, 92, 206, 11, 41].

Despite the lack of an extense literature in EC about these topics, several publications related to Metaheuristics about experimental research can be found. These publications lack a common theoretical background, and they refer similar concepts with different terminology, or use different conceptual frameworks. In the following, we survey the different perspectives used in the literature to refer experimentation, and propose a general framework that summarizes and unifies the terminology about experimental research in EC.

### 2.1.2 Classification of experimental designs

The growing interest in experimental methods in computing has been reflected in an increasing number of publications addressing this topic. As a consequence, the experimental methods reported by the literature in the last years have been enhanced, and at least some of the more obvious pitfalls are generally avoided [26]. Hovewer, this topic has attracted little interest from a purely research perspective, leading to a lack of a shared terminology. In this section we survey this topic, trying to provide a unifying perspective.

In general, we can identify four non-exclusive criteria used in the literature to classify experimental designs, depending on its *objectives*, *problem* that is addressed, the *factor* that is studied, and *other* criteria that does not belong to any of the previous ones. In the following, we review the literature according to this classification. In particular, this topic has been studied with more intensity in Metaheuristics than in EC, so, the following review involves both fields.

A common criteria used by many authors to classify experiments is based on the *objective* that the experimenter wants to accomplish. It is closely related to the difference between engineering and science. Rardin [201] distinguishes between *research* and *development*. The objetive of research is to acquire new knowledge about the algorithm, its behaviour, the relationship between the performance and its components. This perspective is obviously scientific. On the contrary, development relates to an engineering perspective, the goal is to create or use an algorithm that solves a certain problem. Rardin and other authors [69]

maintain the idea that many problems in EC are a consequence of the lack of a clear distinction between research and development. This fact can be observed, for instance, in the strong criticism made by several authors [112, 201] to the high number of publications focused on algorithm comparison instead of understanding why an algorithm performs better than other. Hooker is in an extreme position claiming that this emphasis in competition is "fundamentally antiintelectual" [112].

Eiben also implicitly assumes the difference between research and development in [76], where he states that there are *optimization experiments* and *understanding experiments*. In the same line, but seen from the perspective of parameter tuning, Eiben [78] distinguish between optimization experiments, that set the parameters to achieve the best solution, and understanding experiments, that analyze the dependence between performance and parameters. This perspective is also very close to the one exposed in [11] by Barr, who classifies goals in experimentation as *comparison* and *description*.

The classification in base of the experimental design objective made by Johnson in [122] is more detailed. He assumes that each type of experiment can be associated to one particular type of paper -and, to some extent, assuming that experimentation only has interest for researchers-. He identifies four types of papers (or experiments). The *application paper* is, as its name suggests, a paper whose objective is to apply an algorithm to solve a problem of interest; the *horse race paper* tries to compare two or more algorithms as function of any of their performance measures in a competitive way, the objetive is to claim that algorithm A is better than algorithm B; an *experimental analysis paper*, which tries to better understand an algorithm and finally *experimental average-case paper*, which performs experimentally an average-case analysis of the algorithm when the analytical approach is too complex.

Other authors use the context (or scenario) where the algorithm is run to classify it, or, in other words, the type of problem that the algorithm has to face. Eiben in [79] identifies three scenarios or problems, which are design problems, repetitive problems, and online control problems. This classification can be considered as a refinement of the development scenario used by Rardin [201]. In *design problems*, the practitioner is interested in one solution of the highest quality, once this objective has been satisfied, the algorithm is no needed anymore. For instance, we can identify as design problems classical applications of EAs in engineering, such as antenna design or production chain optimization. On the contrary, *repetitive problems* are those ones where the interest is a sequence of solutions, usually drawn at different time intervals. A classical example of this type of scenario is the optimization of routes of a logistics company [194]. As a special case of a repetitive problem, Eiben identifies the *on-line problems*, which have stronger time constrains, typically because they are used in control tasks where the time dedicated to find a solution is limited. The route selection of a rover might be another example of this type of application. Finally, Eiben also identifies research as an scenario for EAs, however, he does not discuss this class of scenario in his papers.

There are authors that use the factor of study as a criteria to classify experimental designs. Depending on the number of variables considered in the study, Chiarandini in [52] identifies *univariable* and *multivariable* experiments. With the same idea, but different terminology, Rardin uses the tuning method considering whether it uses or not Design of Experiments [201]. If it tunes one parameter each time, she defines the experiment as *sequential*, but if parameters are tuned using Design of Experiments, she names it *factorial design*. On the

Table 2.1: Summary of classification schemes proposed in the literature.

| Author | Publication | Adjectives |
|---|---|---|
| C.C. McGeoch | [165] | Dependency study |
| | | Robustness study |
| | | Proving study |
| A. E. Eiben | [79] | Design |
| | | Repetitive |
| | | Control (particular case of repetitive) |
| J. Derrac | [71] | Single problem analysis |
| | | Multiprogram problem analysis |
| M. Chiarandini | [52] | Univariable |
| | | Multivariable |
| R. L. Rardin | [201] | Scientifical/development |
| | | Design/planning/control |
| | | Sequential/factorial |
| P. R. Cohen | [59] | Exploratory / confirmatory |
| | | Manipulation / observation |
| Several | [59, 165, 59] | Pilot or exploratory |

contrary, Derrac et al. [71] use the number of problems that are introduced in the experimentation, in this way if there is only one problem they name it as *single-problem analysis* and *multiprogram analysis* otherwise.

Finally, several authors use criteria that hardly can be classified in the previous categories. McGeoch, in [165] classifies experimentation in three groups or types of studies, depending on the research question. The first one is the *dependency study*, which is characterized by the interest to discover the relationship between the algorithm parameters and its performance. The second one is the *robustness study*, which tries to characterize statistically the variation of the algorithm properties. Basically, dependency studies use central tendency measures while robustness studies use variability measures. Finally, there are *proving studies*, where the components of the algorithm are studied in relation to their impact in the performance. As Ridge pointed out in [206], this classification has some similarities with the one introduced by Barr in [11], in particular, dependency studies are equivalent to average-case studies as well as proving studies are equivalent to an analysis paper.

Finally, several authors identify pilot or exploratory experiments as a mean to gather a basic understanding about the algorithm [59, 165, 59]. This exploratory experimentation has a limited scope, since its goal is not to gather evidence in order to support any claim. Usually it is carried out in a preliminary stage of the experimentation, and serves to gather basic information needed to design and perform the experimentation. It serves to estimate the computational resources needed by the experiment, identify factors, get a basic understanding about the algorithm and its performance. With all this information, the experiment can be planned and the research questions might be reframed. Ideal exploratory experiments should be limited, using few computational resources and time.

In order to be clear, Table 2.1 shows a summary of the terminology used in the literature

to describe experiments. It should be underlined that, to the authors' knowledge, there have not been attempts to propose formal classification schemes, and the terms showed in the table have been used in an informal way. As it can be seen, many terms refer to the same concept, and different attributes of the experiments are used to describe them, which is confusing. In order to try to clarify this situation, and provide a general framework to contextualize the contributions of the dissertation, the next section tries to provide a general framework to describe experiments.

### 2.1.3 A framework to describe experiments

Given the lack of unified criteria to classify experiments, it is not surprising that this problem is also present in the description of an experiment. Several authors have provided a theoretical background about the components of an experiment, however, there is still a lack of a general framework able to describe the different elements that compose an experiment. In this section we briefly review some literature and use it to propose a framework that systematize the description of an experiment.

In any case, the distinction between the classification of an experiment and its description is far from being clear, and actually, the framework that we propose takes into account elements from the classification schemes previously described. The experimenter motivation, the type of research question, and the objectives of the experiment have a direct impact in how the experiment is designed. And, on the contrary, the result of the experiment might motivate changes in the questions that drive the experiment. There is an interdependence between objectives and design. So, it seems to us as reasonable to take into consideration these motivations (to some extent already reviewed in the previous section) and the elements that build the experimental design. With these precedents, we can propose that the description of an experiment should consider the researcher perspective, and the experiment design as well.

- **Objective**. Which is the objective that motivates the experiment? It might be *research* when the objective is to acquire knowledge; *development* when there is an engineering motivation, where the interest is not acquiring new knowledge but to solve a given problem; *comparison* whether the experimenter is interested to compare two or more algorithms, and finally *exploration*, if it is a preliminary experiment with a limited scope carried out to gather data needed to design the final experiment.

- **Experimental design**. An experimental design is the set of elements needed to plan an experiment in the context of the objectives defined by the researcher. In the context of EC, the elements to define an experiment are the *algorithm*, the *parameter setting* in very likely case that the algorithm is parametizable, and the *problem* instance. In addition, in order to be able to observe the object of study, it is necessary to measure it, and thus, depending on the objective and research question the experimenter has to choose a set of *measures*. The proposed composition of the experimental design is based on the work of some authors. Barr [11] identified three factors in an experiment: the problem, the algorithm, and the test environment. Similarly, Bartz-Beielstein states that an experiment is composed of a problem, an algorithm and a quality criteria [24]. In the same line, Smit distinguishes three design layers corresponding to an application layer, an algorithm layer and a tuning layer [217].

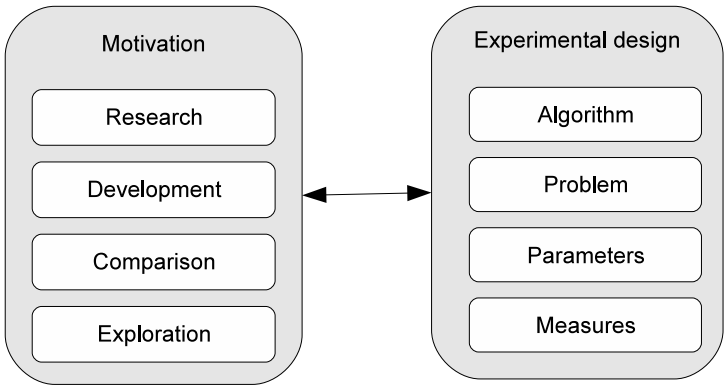


Figure 2.1: Framework for the description of the experiments. It involves two dimensions, the objective that motivates the experimenter and the experimental design, that includes all the elements needed to plan an experiment. These two dimensions are not independent but actually they should be linked.

The objective and the experimental design are different dimensions of experimentation, but they are not uncorrelated. Some objectives are linked more closely to some elements of the experimental design than others. For instance, an engineer interested in solving a certain problem efficiently has a development objective and will be likely interested to find the best algorithm for that class of problem, or, given an algorithm he will try to find the best parameters. A graphical representation of this framework is shown in Fig. 2.1.

In summary, the objectives that motivates the researcher to carry out an experiment, as well as its design are necessary in order to fully describe an experiment. The experimenter objectives might try to answer a research question, to find a solution to a problem, compare algorithms, or gather data in order to design the experiment. Additionally, the experimental design is composed by the elements needed in order to carry out the experiment: an algorithm, a parameter setting, a problem and measures. In the following sections we briefly describe each of the elements that compose the experimental design.

## 2.2 The first component of experimental designs: Algorithm

Metaheuristics are a set of stochastic search algorithms with a wide range of applications. The exact definition of what is exactly a metaheuristic is far from being trivial, and the name might be confusing, since it suggests a similar meaning to metamodel or metaalgorithm [158], which is not the case. Informally we can identify a metaheuristic as an algorithm that sample the search space using some degree of randomness and use the collected information to place new samples, and repeat it until an end condition is satisfied. Metaheuristics have been an intense research topic in AI and there is a large corpus of publications. Several surveys are available in [236, 226, 158, 43, 7].

Metaheuristics is a term that involves a large set of algorithms, and setting a complete

classification is a complex task, due to their diversity and large number of algorithms. Sean Luke divides Metaheuristics in two branches, depending on the number of samples that the algorithm draws on each iteration [158]. So, if only one sample is taken, the algorithm is trajectory-based. This names comes because if the sample points are represented in the search space, they trace a path. On the contrary, if the algorithm draws several points in each iteration, it is a population-based algorithms. A graphical representation of a classification of stochastic search algorithms, with emphasis in Metaheuristics, can be found in Figure 2.2.

It is possible to group population-based Metaheuristics into two sets depending on how new samples are located in the search space. When the algorithm places each new samples based on a limited number of previous samples, it is said to be an EA [79], while if all the samples in the previous iteration of the algorithm influence the allocation of new ones, it is said to be a Swarm Algorithm [42]. The first category is inspired by the Evolution Theory, where natural selection forces an evolution in the population and new individuals inherit their parents characteristics. Similarly, Swarm Algorithms are inspired by natural processes, many of them in swarms or flocks of animals, like social insects [164].

Curiously, from the perspective of the experimental design, all these algorithms can be envisioned as a black box, where an evaluation function is placed, and the algorithm optimizes it with no need of domain knowledge. For this reason, Sean Luke suggests that black-box optimization would be a good name for Metaheuristics. This reason motivates us to provide in the following a broad (and brief) description of Metaheuristics, although the main contributions of this PhD thesis are placed in GP.

### 2.2.1 Trajectory search algorithms

Trajectory search algorithms are a type of Metaheuristic than sample a single point in the search space in each iteration. Depending on how the point is located in each iteration we can distinguish a large number of trajectory search algorithms. This class of algorithms are generally for local search, they are good finding good local maxima, but not so good finding new promising regions in the search space [158]. For this reason this property has been exploited mixing this type of Metaheuristic with more explorative algorithms, usually based on populations. Algorithms that take this hybrid approach have been denominated *memetic algorithms*. The adjective memetic comes from the idea of meme, developed by Richard Dawkins in his book *The selfish gene* [65]. A meme is a unit of cultural transmission, that, in the same way than genes, is able to replicate and transmit itself.

The most simple trajectory search algorithm is *hill-climbing*, which is indeed a well known method in classical AI [210]. It begins by sampling a point at random, then takes another point in the neighborhood of the first one, and, if the new point is fitter, it is selected and the process is repeated, if not, another point in the neighborhood is selected and evaluated. This is a well known local search algorithm, but since it only searches in a limited region, it easily suffers stagnation, which limits the performance of the algorithm. We should point out that hill-climbing is, with the gradient ascent, the most extreme exploitative algorithm, since it only moves to better solutions in the same region that is being explored and the presence of randomness is very restricted.

One of the oldest and best studied trajectory search algorithms, proposed in the eighties, is *Simulated Annealing* (SA) [131]. This algorithm is itself based on an even older algorithm

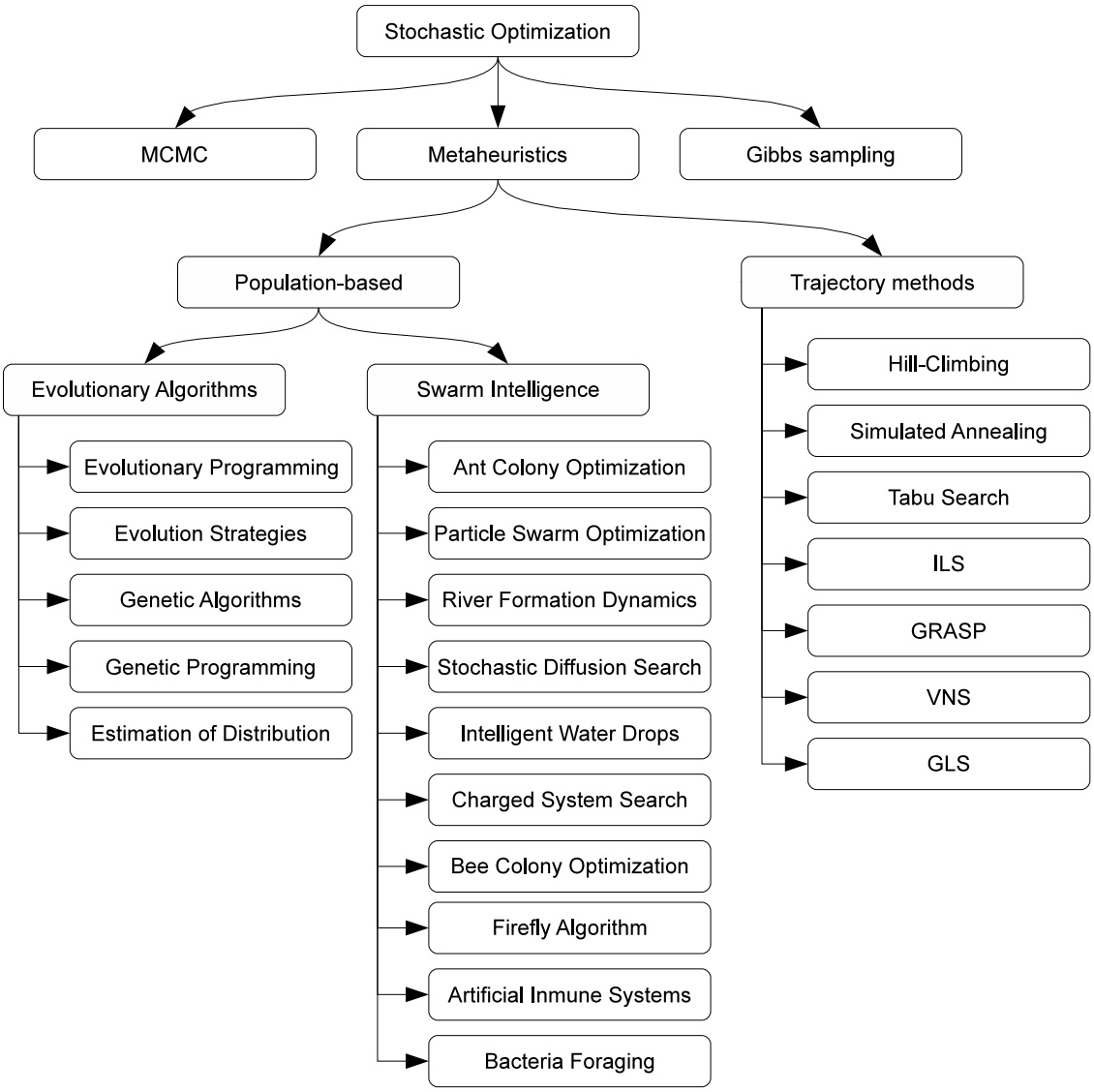


Figure 2.2: Hierarchy of Metaheuristics, including Evolutionary Algorithms, Swarm Intelligence and trajectory methods.

named Metropolis [167]. The idea behind SA is to simulate the process of metal annealing, which consists on controlled cycles of heating and cooling to produce more stable molecular structures in the metal. In practical terms, from a Computer Science perspective, SA is a hill-climbing algorithm that can move towards worse solutions with a certain probability that depends on a parameter named, following its physical inspiration, temperature. The value of the temperature is reduced with time, reducing also the probability of moving to worse solutions. The goal is to benefit exploration in the beginning of the algorithm course, and to reduce it to make the algorithm more exploitative in older stages of the run. Depending on how the temperature is modified, there are a number of variations of the basic SA algorithm.

Another strategy designed to benefit exploration without a significant loose of exploitation is *tabu search* (TS) [93]. To this extend, this algorithm includes memory. The basic idea of TS is to keep a list of the regions already visited without success to avoid visiting them again in a near future, this list named tabu list. Again, depending on the length of the list and how it is managed, there are a large number of variations of the algorithm. This is far from being trivial, the size of the search space might produce huge tabu lists, which requires subconsequently to define a method to manage it.

It was shown that hill-climbing is prone to stagnation, which seriously limits its performance. A more intelligent version of hill-climbing that tries to solve this problem is *Iterated Local Search*, or simply ILS [32]. This algorithm is basically a hill-climbing with random restarts, but these restarts do not begin at random, but using acquired knowledge about the search space. In this way, restarts are more intelligent selecting the starting points. In particular, it keeps the best region of the search space so far found, when the algorithm is restarted, ILS chooses a point in the vicinity of that region, not too far to be within the good region, but not too close, where a local maxima would be reached again. As a result, ILS mixes two types of search, a global search to locate good regions but also a local search to exploit promising regions, for this reason ILS sometimes is mixed with other algorithms, such as a TS of SA.

The list of algorithms belonging to this class of Metaheuristics is extense, some other less known trajectory based methods are Guided Local Search (GLS) [237], Variable Neighborhood Search (VNS) [104] and Greedy Randomized Adaptive Search Procedure (GRASP) [83], among others. Trajectory search algorithms are closely related to the other great branch of Metaheuristics, population-based algorithms, whose most popular family of algorithms are EAs.

### 2.2.2 Evolutionary Algorithms

EAs are a class of population-based metaheuristic inspired by the biological process of Darwinian evolution [79]. Probably, the most characteristic feature of EAs is their capability to combine components of solution candidates though crossover. Unlike other population-based methods, mainly Swarm Algorithms, EAs place new sample points in the search space by combining the information contained in at least two previous sample points selected using a mix of fitness assessment and randomness, simulating the sexual reproduction found in nature. Swarm Algorithms, on the contrary, use the whole population to generate each candidate solution.

Depending on how the candidate solutions are represented at a genotypic level, the type

of genetic operator used, and attending historical reasons as well, there are four big families of EAs: Genetic Algorithms, Genetic Programming, Evolutionary Programming and Evolution Strategies. Recently a new approach to EC named Estimation of Distribution Algorithms (or EDAs) has emerged attracting notable interest, however, this approximation has some particularities that make it hard to classify. In the following, we briefly introduce the two algorithms more strongly linked to this dissertation, and enumerate other EAs to better contextualize this work.

It is important to point out that these branches were created independently by different research groups, and they evolved on their own with different objectives and backgrounds. Only some time after their creation, along the nineties, they begun to be seen as algorithms that share many characteristics, and EC was coined as a name to refer to this field of AI.

#### 2.2.2.1 Genetic Algorithms

Probably the most popular family of EAs is Genetic Algorithms, or GAs, which was first introduced by Holland [110] and then popularized by a set of books, beginning with the one written by Goldberg [95]. A good, but outdated, introduction and survey on GA can be found in [33, 34, 221]. GAs are inspired, like most EAs, in natural selection, and particularly in Genetics. Live beings code all the information needed to build them in form of a sequence of nucleotids, the DNA or RNA. This genetic information is modified though sexual reproduction and/or mutations, which generate diverse individuals. Natural selection operates at an individual layer, selecting the fittests ones. GAs imitates this, coding the candidate solutions in a linear string and simulating sexual reproduction and mutation. Actually, much of the vocabulary used by the GA community comes from this field, terms such as chromosome, gene, locus or epistasis are commonly used in the GA literature. A formal description of GAs can be found in [119].

At a genotypic layer, information is stored in strings named chromosomes. How the chromosomes are coded depends on the particular GA at hand; binary, integer and even float codifications are common; more complex codifications [251], some with strong biological influence [47], are also possible. The way in which the information should be represented in the chromosome generates some controversy [208]. The main genetic operators in GA are mutation and crossover, and they have been also object of controversy [220].

The canonical GA entails a fixed-length chromosome with a binary codification. The reason of the canonical binary representation can be found in Holland's Theorem [110], as described in [9], however this issue is controversial [245], and many authors recommend using the representation that better fits the problem. Holland's Theorem predicts that GAs have an implicit parallelism that is maximized when the number of schemata is maximum, and that happens in a binary codification.

The chromosome is usually divided into chunks or genes that describe one particular characteristic at phenotypic level; however, there is a lack of agreement about the exact meaning of gene in the GA community, and even among biologists [65], and this definition of gene should be handled with care. A gene can take a set of values, each one of these is named allele while the position where the gene is placed in the chromosome is named locus (plural loci). We should underline that in biological systems the function of a gene is not determined by its position in the chromosome. Some proposed GAs imitate this characteristics, for

instance, messy GA [96].

Mutation in the canonical GA is done by just flipping a random bit in the chromosome with a certain probability [110]. The standard crossover uses to be one-point crossover, however there is evidence suggesting that two-point crossover generally yields better performance. Given two chromosomes, one-point crossover selects at random one position in the chromosomes, cuts them in that position and interchanges the remaining chunks.

Of course, more complex codifications are feasible, some even are required. From the perspective of this dissertation, the most important variation of GAs are *variable-length genetic algorithms*, or VLGAs [117, 147], which are used to evolve regular expressions. In this type of GAs the chromosome length is a part of the evolution, and hence it can evolve. The goal is to be able to evolve the complexity of the solution to self-adapt it to the problem, ideally, in increasingly complexity.

#### 2.2.2.2 Genetic Programming

Genetic Programming (GP) involves a wide range of algorithms with diverse strategies and representations, but with a common objective, program induction. As Sean Luke claimed in [158], GP is more a community sharing a research interest on program induction than a coherent set of techniques or research background. The term GP covers many techniques that have little in common, and the research that would be considered common for all these techniques is almost residual. A book about this topic with an excellent review of the literature can be found in [192].

The term GP usually refers to the canonical tree-based GP, proposed originally by John R. Koza in his classical book [136]. In this approximation to GP, the population contains a collection of programs represented by trees, so, in GP terminology, individuals in the population are usually named trees. Selective pressure in canonical GP is introduced by a tournament selection, typically with size seven. As in GA, GP usually uses two genetic operators, crossover and mutation, however, similarly to GA, their role in GP is not clear, and has been an intense area of research for years [161, 121, 244, 140].

The basic tree-based GP algorithm uses unconstrained trees, which in many applications might be a problem. Some programs cannot have any type of node as children of certain nodes, and it would be desirable to introduce some kind of constrain. With this objective in mind, some variations of the basic algorithm have been proposed, for instance, *strongly typed GP*, where the nodes have a type and thus the consistency might be checked [172]. Another approximation is using grammatical constrains [166]. Probably, the best known algorithm based on this perspective is the one introduced by O'Neill in [184], named *Grammatical Evolution*, or GE. This approach uses a variable-length integer linear representation that is used to select a derivation rules from a grammar provided by the user, usually in a Backus-Naur form.

One of the main problems in tree-based GP is its poor locality, which yields a lack of correlation between the genotypic and the fitness landscapes [208]. Consequently, it reduces the capability of the algorithm to exploit the information provided by the landscape, increasing the search difficulty. Another issue that has been criticized in tree-based GP is the storage and manipulation cost of the trees, which is from a computational point of view, high. So, different alternatives have been proposed that do not use trees to represent programs.

One of the major alternative approaches to tree-based GP that does not use tree representations is *linear GP* [10]. Linear GP is inspired by the linear nature of machine code or assembler, where a program is a sequence of instructions that contain the operation and the operators. In this way, linear GP uses a set of registers and operations that are performed with them. The communication among the operations are done though registers, the input of an operation is taken from them, and its output is also stored in the registers.

Another major GP algorithm is *Cartesian GP* [170, 169]. Cartesian GP represents programs as graphs, but at a genotypic layer the graph is coded in a integer linear chromosome. Each element in the graph is represented in the chromosome as a set of integers, one representing the operation type and others representing its position within the graph forming a coordinate, that is why this method is named Cartesian GP. The graph oriented nature of the Cartesian GP makes it easy to represent structures such as circuits.

The list of algorithms that can be classified as GP is extense, and a exhaustive enumeration of them is out of the scope of this chapter. In order to complete the review of algorithms that can be used in the context of Metaheuristics, we introduce in the next section some other major EAs with a looser relation to this dissertation.

#### 2.2.2.3 Other evolutionary algorithms

Another major EC paradigm is Evolution Strategies (ES), proposed by Reichenberg and Schwefel. Curiously, they were working on a problem related to Aeronautical Engineering, wing design, which at first appearance seems far from AI. With this motivation they proposed a self-adapted stochastic optimization method that was named Evolution Strategies. The focus of ES is the numerical optimization in the space of real numbers. The solution is represented by a vector of float values and its main genetic operator is mutation, that is introduced as a gaussian noise. The parameters of the gaussian noise (mean and variance) is modified during the course of the evolution, making ES also an early example of self-adapted EA. A good intro to ES can be found in [35].

The last major EC paradigm, but the first historically speaking, is Evolutionary Programming (EP), proposed by Fogel [87] in the sixties to simulate learning using evolution. In EP, the structure that is evolved is a finite automata, perhaps as a consequence of EP roots in what we could name “classical AI”. Traditionally EP has been used in prediction [79].

A relatively recent paradigm in EC that has emerged strongly is Estimation of Distribution Algorithms (EDAs) [146, 152]. Some authors classify EDAs as a part of GP [192], however they have some unique properties that make them quite singular and difficult to classify. EDAs, instead of keeping a population of solution candidates, take a completely different approach trying to characterize the search space performing a probabilistic estimation of the search space, i.e., assigning probabilities to the search space and sampling the search space according to these probabilities. There are many different proposals of EDAs, however the basic operation is the same. A population of solutions are sampled from the search space according to some probability distribution, then they are evaluated, and, as a result of this evaluation, the probability distribution is updated, providing higher probabilities to the more promising regions. A strong point in pro of EDAs is that the resulting probability distribution provides additional information that can be exploited.

### 2.2.3 Swarm Intelligence

Swarm Intelligence is a recent branch of Metaheuristics that, as its name suggests, is inspired in the emerging intelligent behaviour found in some social insects [164] and animal flocks, although not all the algorithms considered in this category are biologically inspired. In Swarm Intelligence, on the contrary than EAs, individuals in the population exhibit a collective behaviour, each individual affects the whole population and the global behaviour influences individuals. In EAs, one individual only influences directly its offspring, actually, depending on the algorithm design, there would be individuals without offspring, and therefore without influence in the next iteration of the algorithm. The two best known swarm algorithms are Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO).

ACO was proposed by Dorigo [72] and exploits the ability of the biological ants to find good paths between the anthill and the food without a central intelligence or coordination. When an ant finds food, it begins to deposit a trail of pheromones that can be detected by other ants; in that case, the ant follows the trail of pheromones with a certain probability. Despite the ironic lack of ants in ACO [158], the algorithm follows to some extent this strategy though a process of pheromones deposit and evaporation. ACO is just one of the several ant-inspired algorithms such as the Ant System or Min-Max [206].

PSO, on the contrary than ACO, is not inspired in social insect behaviour, but in flocks [128]. The idea behind PSO is to have a population of candidate solutions, or particles using PSO terminology, moving across the search space. This movement is influenced by the position of the best particle so far, and to avoid premature convergence particles have an inertia. In this way a particle in PSO is characterized by its velocity and its position and the resulting behaviour is similar to a flock. PSO is closely related to a recent EA named Differential Evolution (DE) [223, 64].

There is a large amount of algorithms inspired in swarms and nature, moreover, it is a hot research topic and new algorithms are emerging continuously. Some of the latest swarm algorithms are Artificial Bee Colony optimization (ABC) [124], Firefly Algorithm [252], Bacterial Foraging Optimization (BFO) [186], Glowworm Swarm Optimization (GSO) [139] and so on. Even though not strictly swarm algorithms, other nature inspired algorithms are emerging, for instance, Artificial Immune Systems (AIS) [81], River Formation Dynamics (RFD) [197], Intelligent Water Drops algorithm (IWD) [213] or Charged System Search (CSS) [126], just to cite some of them. Another class of algorithms already mentioned are memetic and cultural algorithms, which uses a global search algorithm (typically a population-based algorithm) and a local search algorithm.

So far, an incredibly large number of algorithms have been proposed in Metaheuristics, with very little in common among them. Regardless of the algorithm at hand, all them are run in order to solve a problem. Some metaheuristics were born just to solve one type of problem, others inspired by some natural phenomena. In any case, the relation between algorithm and problem is very close, and an algorithm cannot be studied in isolation, it only takes sense when it is used to solve a problem instance [94]. This issue is addressed in the next section.

## 2.3 The second component of experimental designs: Problem

It is well known that the choice of the problem determines the algorithm performance [246, 94]. When the problem is the object of study it is not a great concern, but when other factors are studied, problem selection becomes a key decision that may bias the results. Designing a fair experimental plan depends on which problems are selected. Despite the key role of this issue and the general agreement about the importance of this topic [111], with some exceptions, not much research has been devoted to this topic.

In this point, it is relevant to introduce two terms generally used in this context. It is necessary to distinguish problem, or problem class, and problem instance [201]. A *problem class* is a generic set of problems with the same statement, but whose numerical values are not specific [201]. There are numerous examples of well known problem classes, for instance the TSP, CSP, 3SAT and so on. Each element in the set of the problem class is a *problem instance*.

In order to assess the performance of an algorithm with a certain problem class, it is necessary to run it with several problem instances, otherwise the results cannot be generalized. How many problem instances are required to be able to claim sound results is not clear, and probably it is one of the weakest points of experimental research [111]. This problem is strongly related to dataset selection in Machine Learning, for this reason, Biratti proposed a strategy inspired by Machine Learning that consists in separating training sets and testing sets [41]. He claims that the problems used for assessing the algorithm's performance should not be used in the algorithm's development.

Problem choice is also a challenging problem from a theoretical perspective. Problem characteristics traditionally used in algorithm analysis fail when they are applied to EAs [192], and therefore, it is needed a new framework able to capture the elements of a problem that can make it difficult to a EA. Modality, separability and regularity are probably more adequate adjectives to describe problem classes in the context of EC. Related to this problem, once these characteristics were identified, how can we create problem classes [76] specific to EC that exploit those characteristics? This question makes more sense under the light of the No Free Lunch Theorem (NFL) [250], which theoretically states that the performance of any algorithm, when it is averaged to all the problems, remains constant. So, under the NFL, an effort to find a superalgorithm with an outstanding performance in all the problems is destined to fail, following that research should be directed to understand which algorithms perform well under which problem characteristics, and provide design guides [99]. Hooker, years before the statement of the NFL, claimed the need of bound algorithm analysis to problem characteristics [111].

Generally, there is a consensus among researchers in this field about how to classify problem classes. Eiben [76] identifies useless, natural and artificial classes of problems. Rardin and Uzsoy [201] take a similar view when they identify four classes of methods to obtain problem instances associated to their datasets: real world datasets, random variants of real datasets, published and online libraries and finally randomly generated instances. Similarly, Bartz-Beielstein in [25] distinguish three types of problems: test functions, real-world optimization problems and randomly generated test problems. Table 2.2 provides a summary of these problem classes.

Following Bartz-Beielstein, we identify three groups of problems to test EAs: test suites,

Table 2.2: Summary of problem classification schemata proposed in the literature.

| Author | Publication | Classes of problem |
|---|---|---|
| A. E. Eiben | [76] | Useless |
| | | Natural |
| | | Artificial |
| R. L. Rardin | [201] | Real world datasets |
| | | Random variants of real datasets |
| | | Published libraries |
| | | Randomly generated instances |
| T. Bartz-Beielstein | [25] | Test functions |
| | | Real-world problems |
| | | Randomly generated test problems |

that contain a fixed number of problem instances; instance generators, that create randomly problem instances and finally real world problems, that, like its name suggests, are problem obtained from the real world. A good discusion about advantages and disadvantages of using synthetic or natural problems can be found in [94]. The next sections are dedicated to describe some widely used problem suites. Due to its special interest in the context of this dissertation, we pay more attention to test suites used in GP.

### 2.3.1 Test suites

Test suites are public collections of selected problems used to analyze algorithms. Usually, test suites are used to assess the performance of several algorithms and compare them in order to determine which one has the best performance. The utility of test suites in EC is double, on the one hand they provide a set of common problems, enabling the comparison of the results among different studies. On the other hand, ideally test suites are designed in order to assess some attributes of the algorithms under study, and then fine-grained understanding about their performance can be more easily achieved. However, which properties should have test suites in EC is still an open problem [193].

Usually, test suites have a strong bias to numerical optimization, and these tests are stated as a numerical maximization (or minimization) of a certain expression. One remarkable advantage of this approach is that the maximum (or minimum) of the problem can be known in advance, and therefore the performance of the algorithm is easily assessed. Another advantage is the existence of parametriced test suites, so some property of the problem is tunable in such a way that the experimenter can modify, for instance, the level of difficulty or other characteristic of the problem at will. This is a valuable feature in many experimental designs, and can be an important help in order to understand and assess the algorithm.

A good example of test suite is the first one in EC [23], that was proposed by De Jong in his PhD dissertation [67]. He proposed five functions for $n$-dimensional real-number optimization, each one of thes selected according to some special feature that made it interesting to evaluate. Their definition is shown in Table 2.3, while Figure 2.3 shows the shape of De Jong's functions in the bidimensional case. It is interesting to underline that although De

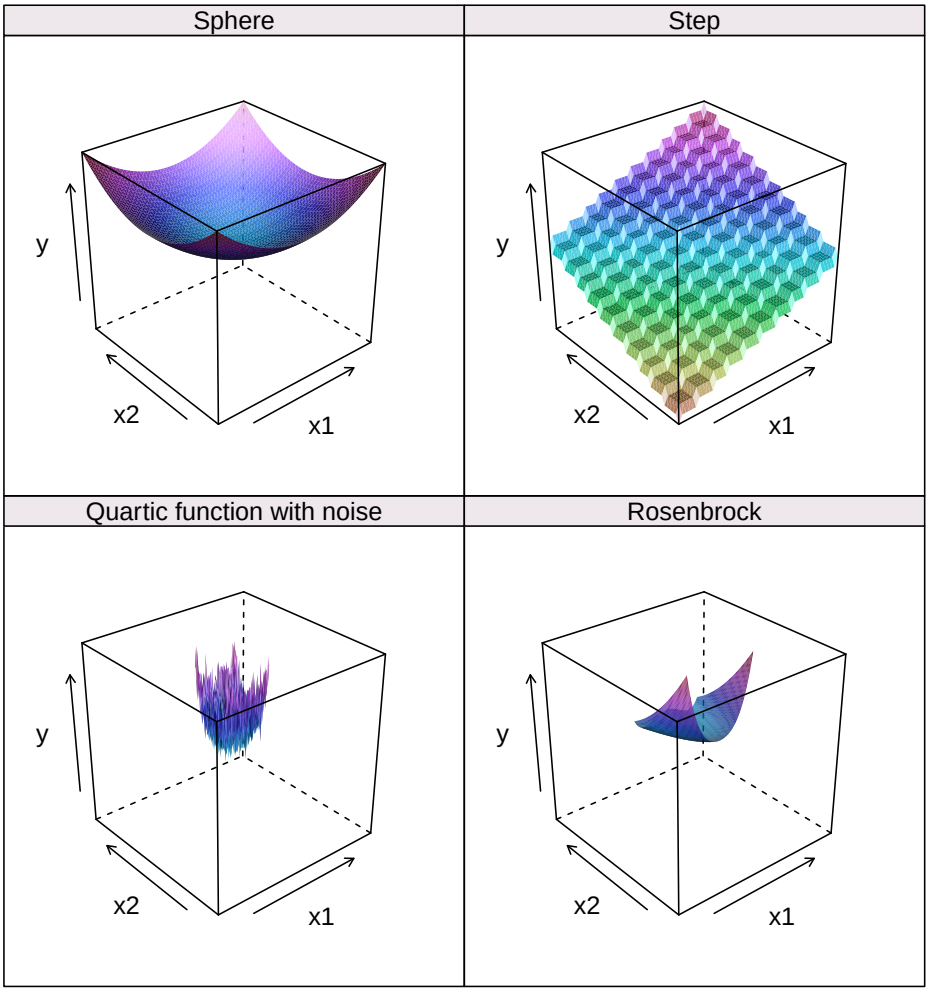


Figure 2.3: Plots of the bidimensional version of four or the five De Jong functions test suite: Sphere, Rosenbrock, step, and quartic function with noise.

Table 2.3: Analytical definition of five De Jong functions.

| Function | Expression | Domain | Minimum |
|---|---|---|---|
| Sphere | $f_1(\overline{x}) = \sum_{i=1}^{n} x_i^2$ | $\mid x_i \mid \leq 5.12$ | $f(0,\ldots,0) = 0$ |
| Rosenbrock | $f_2(\overline{x}) = \sum_{i=1}^{n-1}((1-x_i)^2 + 100(x_{i+1} - x_i^2)^2)$ | $\mid x_i \mid \leq 2.048$ | $f(1,\ldots,1) = 0$ |
| Step | $f_3(\overline{x}) = 25 + \sum_{i=1}^{n} \lfloor x_i \rfloor$ | $\mid x_i \mid \leq 5.12$ | $f(([-5.12,-5),\ldots,$ $[-512,-5))) = 0$ |
| Quartic | $f_4(\overline{x}) = \sum_{i=1}^{n}(ix_i^4) + N(0,1)$ | $\mid x_i \mid \leq 1.28$ | $f(0,\ldots,0) = 0$ |
| Sheckel (2D) | $f(x_1,x_2) = \frac{1}{0.02+\sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ji})^6}}$ | $\mid x_i \mid \leq 65.536$ | $f(-32,-32) = 1$ |

Jong functions have had a strong influence in EC research, and they have been widely used, De Jong now advocates against his own test suite [158]. A brief discussion about these five functions follows:

1. **Sphere**. This is a simple and smooth unimodal function whose minimum should be easy to find.

2. **Rosenbrock**. Also known as Rosenbrock's banana due to its shape in the bivariable representation. In this case, it forms a valley around a hill where algorithms use to stagnate in a local minima. So, this function evaluates the capability of the algorithm to reach the global optimum in presence of local minimal and a smooth landscape.

3. **Step**. This function is characterized by the presence of a high number of plateaus that difficulties the search process due to the lack of exploitable information in these regions.

4. **Quartic function with noise**. This function represents a rather simple surface that is roughed with a gaussian noise that makes difficult the search process. This function assesses the behaviour of the algorithms when it is used with a noisy fitness function.

5. **Sheckel's Foxholes**. Extreme problem with a plateau that presents several steep peaks with many local minima. The coefficients $a_{ji}$ shown in Table 2.3 are given by:

$$a_{j1} = \{32, 16, 0, 16, 32, 32, 16, 0, 16, 32, 32, 16, 0, 16, 32, 32, 16, 0, 16, 32, 32, 16, 0, 16, 32\}$$
$$a_{j2} = \{32, 32, 32, 32, 32, 16, 16, 16, 16, 16, 0, 0, 0, 0, 0, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32\}$$

We should mention that many test suites have been proposed, for instance, by Fogel [86], Schwefel [212] and Eiben [74], which include some of De Jong's functions. In addition to these test suites, in the last years several test-suites have emerged under the shelter of competitions holded in the two main EC events, CEC and GECCO, and some publications [106].

Probably the most used suite case [185] is the one used in the CEC 2005 Competition on Real-Parameter Optimization proposed by Sugan [225]. Another CEC competition is the Competition on Large Scale Global Optimization; in the 2010 edition [227, 228] the test-suites were composed by $25$ functions, $5$ unimodal and $20$ multimodal separated in several categories: basic, expanded and hybrid functions. Most of these functions, $22$, are non-separable while $2$ are completely separable and one is separable near the optimum. Tang defines a separable function as a function whose maximum or minimum can be solved as the sum of the minimum of several functions of one variable. The test-suites used by CEC competitions are probably the most used ones by the research community. Similarly, GECCO has also hold several competitions on real-number optimization, among others. GECCO 2009 Black Box Optimization Benchmarking, and subsequent editions, provided noisy and noise-free functions. The noiseless functions contained $24$ functions [84, 103].

Test-suites so far described are focused in numerical optimization. Nonetheless, sometimes the object of study is not the algorithm, but rather the problem for its practical or academic interest. Usually, these problem classes have attracted a large amount of research, problems such as the TSP, Knapsack or 3SAT are good examples of this. In order to achieve comparable research results on this problems, several specific test-suites have been developed. For instance, we can mention the TSPLIB [204], which is almost of universal usage

in research related to the TSP, the UCI Machine Learning Repository datasets for Machine Learning applications, or the Tomita test-suite for language induction [229].

There are also specialized domains in EC where classical test-suites are not applicable. A good example might be multiobjective optimization [133, 192, 79]. Examples of specialized test suites are constrained real-parameter optimization [149, 162], dynamic optimization or multiobjective evolutionary algorithms. Many of these problem-specific test-suites are provided in form of instance generators.

### 2.3.2 Instance generators

Instances generators take another perspective. If test-suites contain a collection of fixed problem instances, instances generator create new problem instances that belong to a certain problem class. Instance generators are not exclusive of EC, but rather they have a long tradition in AI, and there are available a long list of instance generator for almost all the most important AI problems, such as the CSP, SAT, or DFA induction [142]. A good repository of test instances is kept by Spears in [219].

Using instance generators has several advantages. Firstly, since an arbitrary number of problem instances may be created, some problems associated to the limitation of the available problem instances are solved. As a result, the scale of the experimentation can be increased and open new experimental approaches. And secondly, many instance generators are parametizable, and thus they can generate problem instances with different properties like the problem size or difficulty. It allows to manipulate the algorithm environment in order to gain more control, and therefore, design better experiments that otherwise could be difficult, if not impossible, to implement.

In the specific context of EC, instance generators have been closely related to the fitness landscape [222, 208]. This concept involves a geometric interpretation of the fitness space associated to an idea of neighborhood. Fitness landscapes have attracted much research interest because they have served as a basis to characterize problems and algorithms. A hot research topic is the measurement of problem difficulty, which is close related to fitness landscapes [233, 193]. A particular instance generator that has had a notable impact on EC research, or more specifically in GA, is the NK-landscapes [4]. This instance generator creates problem instances with size $n$ and $k$ epistatic interactions, which means that its size and difficulty can be tunable.

So far, we have discussed the role of problem instances and instance generators. In particular, they copy the characteristics of some type of problem, such as numerical optimization, or the TSP, to ease experimentation. We have seen that certain types of algorithms require specific test-suites, multiobjective algorithms, for instance, require specific problem instances suitable to exploit their special properties. There are, however, some fields in EC that, due to historical reasons rather than to technical ones, have been using their own test suites. One clear example of this is GP. Due to its importance in the research reported in this dissertation, we discuss this topic in more detail.

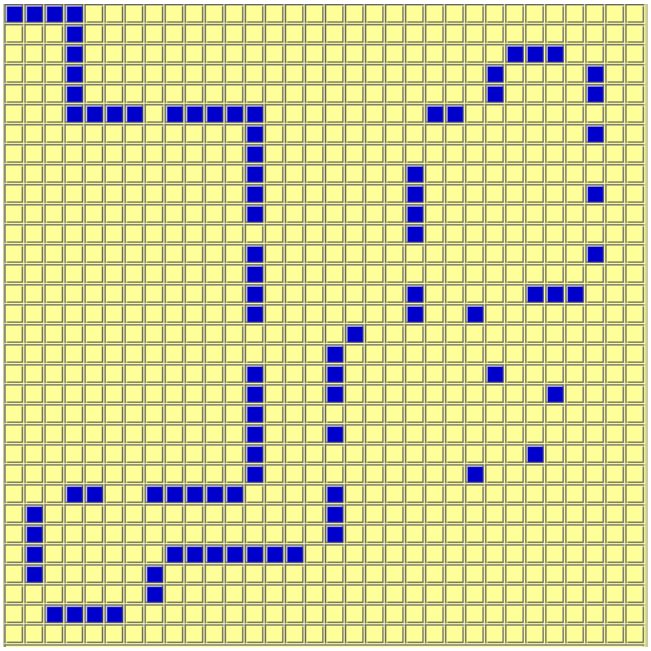

Figure 2.4: The Santa Fe trail, which is used in the artificial ant problem, one of the most popular GP test problems proposed by Koza (source: Wikipedia).

### 2.3.3 Test suites in Genetic Programming

GP, in relation to the rest suites used in EC, has, to some extent, some particularities that make it different. Many of the test suites used in the GP literature do not use the test suites that are common in the rest of EAs, but rather it is common to find some test problems that are rarely found outside of the GP community. One possible reason can be found in the type of problem that is faced by this type of algorithms, but more likely it is due to historical reasons, and the influence of John Koza in this field.

There are a set of widely used test problems in GP, those proposed by Koza in his first book [136]. Despite the lack of solid theoretical foundations to choose those problems, Koza tried to represent several types of problems solvable by GP, for instance, path finding or boolean problems. Traditionally, these problems have been widely used by the GP literature, and some of them play a key role in this PhD thesis. The four test problems proposed by Koza that play a major role in this dissertation are described in the following.

1. **Artificial ant**. This problem deals with the simulation of an ant placed on a grid. The ant has to move collecting food lying along a trail. There are some trails in the literature with different grid sizes, however, the most used one is the Santa Fe trail, which is a 32x32 toroidal grid with 89 food pellets, as depicted in Figure 2.4. The fitness is the count of pieces of food found by the ant before it performs a given number of steps. Koza reported in his book a timestep value of 400, however, there is a strong evidence suggesting that this value is a typo in the book, and real maximum number of timesteps used by Koza is 600 [215]. It is said that this problem is hard and the solution

is constructed almost randomly [143], nonetheless Christensen was able to solve this problem in $20,696$ evaluations [55]. One interesting point about this problem observed by Sean Luke [160], is that many different individuals from a genotypic perspective may have the same fitness.

2. **k-multiplexer**. The goal of the k-multiplexer is the design of a boolean function that implements a k-lines multiplexer. The interpretation of the boolean function might vary from a program to a digital circuit. The difficulty of this problem depends on k, obviously, higher k values yields more difficult problems. Typical values of k are $6$ and $11$. Luke observed [155] that the 6-multiplexer generates many inviable code in comparison to the 11-multiplexer and a high probability of neutral crossover.

3. **k-parity**. This is, as the k-multiplexer, a boolean problem whose goal is to design a boolean function. There are two versions of the k-parity, even and odd. An even-parity function of k lines returns true if its argument has even number of bits. The odd-parity function, on the contrary, return true if there is an odd number of bits. As the k-multiplexer problem, difficulty increases with k.

4. **Linear regression**. Regression is a classical problem in Mathematics, and one of the problem domains where GP is more popular. Basically, regression deals with, given a set of n-dimensional points, finding a function $f(\bar{x})$ that fits well the given points. Regression might deal with non-linear models, however, linear regression is widely used as a test problem for its simplicity. In particular, Koza proposed to fit a quartic polynomial $x^4 + x^3 + x^2 + x + 1$ given $20$ points sampled in the domain $[-1, 1]$. Luke observed that this problem generates more inviable code in comparison to the two previous boolean problems. The explanation that he offers is given by the existence of ratios or products with infinite numbers or NaNs, and decimation (a very big number that masks another smaller one) [156, 160].

So far, we have described two elements needed in order to carry out an experiment: an algorithm and a problem. However, almost any algorithm, depends on a set of parameters, and its behaviour may rely strongly on them. Hence, it makes sense to claim that any rigorous experimental design must take into account this fact, and not consider the algorithm and problem in isolation w.r.t. the algorithm parameters. A well defined criteria should be defined in order to select those parameters. This claim is true specially for Metaheuristics in general, and EC in particular, since they tend to be algorithms with a high number of parameters. The next section is dedicated to discuss this important, and often forgotten, topic.

## 2.4 The third component of experimental designs: Parameters

Parameter selection is one of the most challenging and studied problems in experimental evaluation of EAs, and probably it is also often ignored in the experimental designs found in the literature. It is rather common to find papers (including several written by us) that report in detail some algorithm A, and compare it to another algorithm B, claiming that A is better than B. The number of methodological pitfalls with this practice is large, and, despite the extensive literature that warns us against this practice [112, 11, 76, 41], is it still common

practice. One of the most important pitfalls is that the algorithm parameters have been set *ad hoc*, or the method used to tune the parameters is not reported.

It is well known that, depending on the parameter choice, the performance of an algorithm may vary in orders of magnitude [8, 187, 78]. Quite often papers only report details about the algorithm, and not about how the algorithm was configured. Due to the major influence of parameter setting on the performance, it is possible to better tune algorithm A than B, and as a consequence experiments will show that A outperforms B. The experimenter might be tempted to incorrectly conclude that A is better than B. This is, unfortunately quite common. Much research have generated biased results by devoting more effort to search good parameters for one algorithm than another. This relationship between algorithms and parameters has lead Biratti to claim that "the configuracion procedure becomes an inseparable part of the algorithm" [41], consequently, a balanced and fair amount of effort should be dedicated to configure the algorithms under study [12].

Despite this methodological concern, there are some other notable problems with the configuration of parameters. A general practice is to look for the optimum parameters and then assess the algorithm, but the concept of optimum parameters is questionable. First, what does optimum parameters mean? The NFL theorem limits the scope of what we can expect from an algorithm. An algorithm may achieve an outstanding performance on a single problem instance, or an algorithm with a reasonable performance on a problem class or classes. In other words, the same algorithm might be specialist or generalist [78] depending on the algorithm configuration. But even in case that the configuration had been chosen to generate an specialist or generalist algorithm, the optimum configuration changes during the course of the run [75], and depends on which performance measure is used [78]. It is interesting to note that automatically tuned parameters are usually rather different from those selected using the experience, rules-of-thumb, common sense, or other *ad-hoc* methods [217].

Due to the importance of this topic, it is not surprising the notable extension of the literature about this issue despite the complains of some authors about the lack of interest in this area [78]. Subsequently, many different methods to configure an algorithm have been proposed, and classifications of these methods. Examples of authors that have proposed criteria to classify those methods are Angeline [5] or Hinterding [108]. Following Eiben [75, 78], it is possible to identify two strategies to address this problem, parameter control and parameter tuning. *Parameter tuning* deals with the selection of static values of the parameters, while *parameter control* deals with methods to self-adapt parameters during the course of the run. We provide more details about these methods in the next subsections.

### 2.4.1 Parameter control

Parameter control deals with algorithms that are able to self-adapt their parameters. The idea behind parameter control is using feedback to adapt the algorithm parameters. This is not a new idea, actually it has been around EC from its beginning. For instance, traditionally ES has used the 1/5 rule to adapt the mutation rate [202], which is itself a self-adaptation mechanism. This research line is still a hot topic, and there is an intense activity on it [97].

There are some solid reasons to use parameter control. As Eiben notes [75], an EA is a dynamic adaptive process, and thus, using a fixed set of parameters contradicts this spirit. Using an algorithm that dynamically adapts its parameters is more respectful with the nature

of EAs. There are also more practical reasons to use parameter control techniques. The problem of finding good parameter values is itself a search problem in the parameter space, which is usually much larger than the solution space, so, finding optimal parameter values is rather unlikely. Fortunately, it seems that suboptimal controlled parameters performs better than suboptimal selection of fixed parameters [75]. In addition, it is well known that the optimum equilibrium between explotation and exploration is dynamic, and thus, the optimal parameter configuration changes along the course of the run [43, 34], so, using fixed parameters seems clearly suboptimal.

Eiben in [75] proposed a bidimensional classification scheme for parameter control based on what component of the algorithm is changed and the type of control that is performed. Despite the difficulty of enumerating all the components of an EA that can be controlled, Eiben enumerates the following *components*: Representation of individuals, fitness function, variation operators and their probabilities, selection operators, replacement operators and population. More interestingly, the criteria of type of control deals with the method used to decide how to change parameters. Eiben identifies three types of *control methods*.

1. **Deterministic parameter control**. The control is performed by a set of heuristics that take deterministic decisions. Given the same conditions, with the exception of the random seed, these techniques will take the same decisions. Those control methods that use feedback from the algorithms are excluded from this category. It is interesting to note that some Metaheuristics belong naturally to this category. For instance, the basic algorithm of SA modifies its main parameter, the temperature, as a deterministic function of time, which can be considered as a deterministic parameter control.

2. **Adaptive parameter control**. It uses feedback from the algorithm and some heuristic to set the parameters. This is the case, for instance, of the 1/5 rule usually used in ES.

3. **Self-adaptive parameter control**. The parameters of the algorithm are encoded within the individuals, and therefore they evolve. In this case, the EA not only searches to optimize the fitness function, but also the parameter settings. This type of control method is the one used in metaevolutionary algorithms, which is an active research area.

In general, using parameter control techniques notoriously improve the algorithm, even when the control method could be better. The point that is perhaps more used against parameter control is that these techniques provide little or none information about how the parameters affect the algorithm behaviour. It is a serious concern in some scientific contexts when the objective of the researcher is to acquire knowledge about the algorithm [24]. This problem is solved, at least partially, by parameter tuning.

### 2.4.2 Parameter tuning

Parameter tuning deals with the problem of choosing the parameters that are set before the algorithm is run and remain fixed along the run. Usually *ad-hoc* methods are used to tune parameters. The most simple one is just using the default parameters found in the algorithm or its implementation. This is the case, for instance, of tree-based GP, where the parameters proposed by Koza are widely used even when they can be markedly improved in many cases.

Well based methods to tune parameters are needed, at least, for two reasons, one practical and another theoretical.

Parameter tuning is important from a practical point of view. The difference between default parameters and customized ones can be of orders of magnitude. This result has been widely reported by several authors, perhaps one of the most complete studies performed by Pellegrini [188], who compared default and tuned parameters in five Metaheuristics (TS, SA, GA, ITS and ACO), finding strong differences in the performance. Of course, it could be argued that parameter tuning is a time-consuming task, and some times this additional effort is not worthy. Other authors counterargument that, due to the availability of several tuning methods that are quite straightforward to implement, and the performance improvement, not using parameter tuning does not admit any excuse [78, 25].

But there are also theoretical reasons to use parameter tuning. First, parameter tuning is intrinsic to EAs design, a fair comparison between two algorithm requires a description of the algorithms, but the method used to tune the algorithm parameters as well [41]. Secondly, parameter tuning might provide valuable information about the algorithm internals. As Eiben noticed [78], one might take two approaches: configure an EA optimizing its parameters, or analyze an EA studying the dependence between its performance and its parameters. This is one strong point in favor of parameter tuning in comparison to control methods, it provides valuable information about the algorithm that could be exploited.

So far, it is not surprising that this issue has attracted notable research. Probably, the most prolific author in this area is Agoston E. Eiben. He classifies parameter tuning methods in three categories, depending on the strategy used to save runs [78]. It might try to reduce the number of parameters to optimize (some authors refer to this as screening), reduce the number of tests, or both. There is also a fourth category related to reducing the number of function evaluations, however, Eiben does not know anyone that had used this strategy so far.

In the following, we will use a classification scheme inspired by the one proposed by Ridge [206], and briefly summarize some relevant literature. In this classification scheme, we distinguish between analytical, automated and empirical approaches.

#### 2.4.2.1 Analytical approach

Analytical approaches try to deduce formal models, and use them to determine good parameter settings. This approach has, however, some troubles. For instance, Ridge [206] concludes that the state-of-the-art of analytical approaches is not ready to address this problem. Actually, this is a general complain made by researchers about theoretical research, theoretical results are rarely exploitable in practice [193]. A good example of this approach to the representation problem [208] is given by Holland's Theorem. This theorem predicts that the implicit parallelism found in GA is maximized when the codification contains the maximum number of schemata, which is obtained with a binary representation. However, this approach has been widely criticized since it depends on a large number of assumptions that are not usually found in practice [203, 245].

#### 2.4.2.2 Automated approach

The second approach to parameter tuning is automated approach. Parameter tuning is, ironically, a search problem in the parameter search space, hence not surprisingly, classical search methods, including stochastic and not stochastic methods, might be used [78]. Automated approaches are, following Ridge, composed by two subcategories, self-tuning, which corresponds to Eiben's control methods, and heuristic tuning, which use search techniques to tune the parameters before the algorithm is run. Within this category, several approaches can be taken. For instance, a metaheuristic such as a GA, can be used to search the parameters. Perhaps, the main drawback with this strategy is the difficulty of interpreting the results in order to achieve an understanding of the algorithm behaviour. Additionally, there is to some extent a recursive problem. If we run a metaheuristic to search the parameters, we need to tune the metaheuristic as well. Other search techniques might be also used [25, Chapter 6].

A radically different approach is taken by Birattari [37], who applied Machine Learning techniques to tune parameters. He applies a Machine Learning algorithm family generally known as race algorithms [163], that, given a limited set of evaluations, try to obtain the best parameters. All these algorithms share the same overall operation. Given a set of candidate parameter configuration, they are tested a given number of times, and those that are not statistically significant in comparison with the best configuration found, are removed and the process is repeated. Depending on the type of statistical test used, different racing algorithms have been proposed. Probably, the most popular one is F-Race [39, 36], which is based on a nonparametric Friedman's two-way analysis of variance by ranks. A survey about this topic can be found in [40].

#### 2.4.2.3 Empirical approach

The third subcategory proposed by Ridge are empirical approaches, which are characterized by the use of empirical models that relate the configuration and the algorithm performance. The model itself is a valuable outcome of this approach since it provides information about how the algorithm performance is influenced by the parameter configuration. In this approach, the algorithm has to be run several times with a controlled environment, then the algorithm performance is statistically analyzed and modeled.

The most simple, widely used, and perhaps also the most mediocre method, is what Ridge names *one-factor-at-time* (OFAT) [206]. This is usually an *ad-hoc* method due to its lack of conceptual complexity. OFAT involves the repetition of the runs in unifactorial experiments, where only one factor (parameter in EC terminology), is changed each time. The list of drawbacks is extense [174, Chapter 7]. Since only one factor is considered each time, it is not possible to identify and model interactions between the factors. Additionally, the points in the parameter space are not optimally sampled, which yields more runs than the strictly needed; this method is time consuming and parameter configuration obtained are rarely optimal [75].

This is actually a well studied problem in Statistics and partially solved with the *Design of Experiments*, or DOE [173, 59]. It has been widely applied with notable success in several domains, such as civil and chemical engineering. DOE is a multifactorial statistical technique that places, using solid statistical criteria, sample points in the search space in order to be able to identify iterations among the factors while lowering the number of samples needed to

draw statically sound conclusions. Nonetheless, there is a notable corpus of literature using DOE to study numerous algorithms such as GA [177, 203], PSO [29] or ACO [206]. A detailed discussion about the use of DOE in EC can be found in [25].

The main drawback of using DOE is its poor scalability when the number of factors to analyze is increased. Another serious problem with DOE is that it only can, in general, analyze a tiny region of the parameter space. Let us suppose, for instance, a standard GP algorithm with, being conservative, ten factors and two levels for each factor. Then, a classical $2^k$ *full factorial design* would yield $2^{10}$ evaluation points. Due to the stochastical nature of the performance measure, it would require several runs for each evaluation point. Let us suppose that each point is evaluated $20$ times, then DOE would require $20480$ runs. Even this conservative example, we find a notable need of computational resources. For this reasons, there are several alternatives to the basic full factorial design, for instance, the $2^{k-p}$ *fractional factorial design*. Probably, one of the strongest points in favor of DOE is the statistically significant information provided about the influence of each factor and their iterations to the algorithm performance. It is useful to select which parameters should be studied. Another method related to DOE is the one proposed by Adenso-Diaz in [1], named CALIBRA. He applied Taguchi's DOE with a further local search.

One problem of DOE is that it places the sample points before the experimentation begins, so, information gathered during the experimentation is not used as feedback to improve the search. Sequential experimental designs try to solve this problem using feedback during the course of the experimentation. One of the best known is *Sequential Parameter Optimization*, or SPO, proposed by Thomas Bartz-Beielstein [28] and implemented in SPOT (Sequential Parameter Optimization Toolbox) [27]. SPO is an iterative method that builds a metamodel based on the observations made so far, and allocates new sample points in the parameter space in base of that metamodel. Then, a new set of runs are carried out using the new parameters and the metamodel is updated. This process is repeated until the budget of computational resources are exhausted. An hybrid algorithm between SPO and F-Race was proposed in [30].

Another tuning method named *Relevance Estimation and VAlue Calibration* (REVAC) was introduced by Nannen and Eiben in [178]. A detailed discussion about this method can be found in [216]. REVAC is related to EDAs to some extent, it builds for each parameter an utility distribution, giving higher probabilities to those values of the parameters that are likely to increase the performance. There are two mayor problems with REVAC. One is that it cannot handle categorical parameters, i.e., it is limited to numerical parameters in contrast to SPO. The other problem is that it does not consider iterations among the factors.

Once the algorithm, the problem, and the parameter configuration have been set, the experiment can be run. However, in order to be useful to the researcher, the course of the experiment should be observed, in other words, it is necessary to collect data about the experiment. We have to measure the experiment and therefore we need measures.

## 2.5 The fourth component of experimental designs: Measures

In order to be able to observe what is happening in the course of a run, it is necessary to take measures. One of the intrinsic characteristics of EC is their complex dynamics, that generate

large amounts of information, and therefore collecting all the information would require large storage and processing capabilities, which in practice limits seriously what information can be stored. Subsequently, depending on the experiment motivations, measures have to be selected [165], which is not always a trivial task. In Barr's words, "perhaps, the most important decision one makes in an experimental study of heuristics is the definition, or characterization, of algorithm performance" [11].

Due to the variability of potential research questions, and the intrinsic complexity of EAs, it is difficult to find generally accepted measures [23, 24]. Nonetheless, there are some measures such as the fitness or success probability that are easy to find in the literature, and some authors have proposed some guidelines to help with the decision of which measures should be used. For instance, Hooker in [112] suggests measuring only those variables predicted by the model, given that there were a model of the algorithm, which is not the general situation in EC. In any case, it cannot be said that there is a shared criteria to select measures, the decision usually depends on the personal experience of the experiment designer.

Even in case that there were a consensus about which measures should be used w.r.t. the researcher motivations, there are some debate about how they should be used [193]. Eiben et al. [76, 79] defend that, depending on the goal, we should observe peak or average values. It is interesting to observe this reasoning in the context of the experiment classification seen in section 2.1.2. Design domains look for extreme behaviors because they are interested in the best solution, which is, by definition, an extreme case. On the contrary, in repetitive domains, average behaviors are more interesting since they involve the whole population, and thus it is more likely to find a solution in successive runs. Birattari, on the contrary, claims that the best fitness -an example of extreme behaviour discussed by Eiben- is a biased estimator, and thus should not be used [38].

A common practice in EC is to execute an algorithm $N$ times and keep the best individual. Birattari criticizes this practice by arguing that this practice is actually a restart, which is a particular practice in Metaheuristics, and therefore experimentation using this method is no longer testing a given algorithm, but that algorithm with restarts [38]. In other words, the evaluated algorithm is being changed by the experiment. Although the Birattari's argument is convincing, one could counterargument that many times, the restart in the EA is introduced not just to find the best solution, but rather to obtain metrics that require several samples, for instance the success rate, or to determine the statistical properties of the measure.

Due to the complex nature of EAs, it is not surprising that there are many measures that can be collected. The criteria that should be used to select some instead of others is still unclear. Having a general knowledge of which measures exist might help in this task, so it is interesting to classify measures in EC, and at the same time might help to place the main object of study of this dissertation. In the next section we review some classification schemes informally proposed by the literature.

### 2.5.1 Classification of measures

Despite there has not been, to the authors' knowledge, any attempt to formally classify measures, several authors have informally introduced some measures classification schemes, most of them only consider performance measures. Almost all the authors interested in this topic mention the quality of the solution and the amount of resources needed to reach a

solution, indeed these are the two sides of performance measures: the quality of the solutions and the cost of getting them. Nonetheless, terms used in the literature are not uniform, and sometimes they represent slightly different semantics that may produce misunderstandings. We should mention that, even tough performance measures are the most widely used, there are several ones that cannot be considered in this group.

Not surprising, all the reviewed authors identify the quality of the solution as a class of measures. However, the exact meaning of quality of the solution is sometimes unclear. Solution quality might be identified as a synonymous of individual fit, which would be incorrect. As Rand advises [200], the solution quality is about the goodness of a solution, while the fitness is used in the selection phase during the course of the EA. So, solution quality and fitness might use different elements. To illustrate this point, let us consider two solutions given by an EA with the same fit, but with different sizes. Usually smaller individuals are preferred to larger ones because the resulting systems are easier to implement and interpret, so, the quality of the small solution is higher than the large one, even when their fit is the same. Nonetheless, the distinction between solution quality and fitness is rather tricky and several times they are used interchangeably.

The most general classification scheme so far found in the literature was proposed by Burke [48], who, depending on which search space the measure is related to, distinguishes genotypic and phenotypic measures. *Genotypic measures* are those measures that consider any characteristic at genotypic level, i.e., the structures used by the algorithm to represent the individual; similarly, *phenotypic measures* are those ones taken in the fitness space. So genotypic measures are not affected by the fit of the individuals, but rather by their representation within the algorithm and serve to know how the population is. Examples of genotypic measures are diversity [48], or individual size. On the contrary, phenotypic measures are not directly affected by the representation of the solutions, but by their fit. For this reason, from our point of view, phenotypic measures can be identified with performance measures.

So far, we can identify performance measures and genotypic measures. In addition, we will consider a third group of measures, that we will name *specialized measures*. These measures only make sense in the context of a certain problems or algorithms, due to its nature or historical reasons, providing information about the algorithm, problem or solution that typically do not make sense out of that scenario. For instance, measuring how fast an algorithm can adapt its population to changes in the solution makes sense in dynamic optimization, but not in more classical scenarios. A graphical representation of this classification scheme can be found in 2.5. Each category in the proposed classification is introduced and discussed in more detail in the following subsections.

### 2.5.2 Performance measures

The most widely used measures are those that estimate the goodness of an algorithm in terms of solution quality and resources consumed by the algorithm. Both terms, solution quality and resources, should be understood in a broad sense. Not surprisingly, this type of measures have been described more in detail in the literature, and some classification schemes have been proposed. In general, most of the consulted authors agree in distinguishing between solution quality and the resources consumed to get the solution. But there is a lack of agreement in the terminology, sometimes with slightly different meanings.

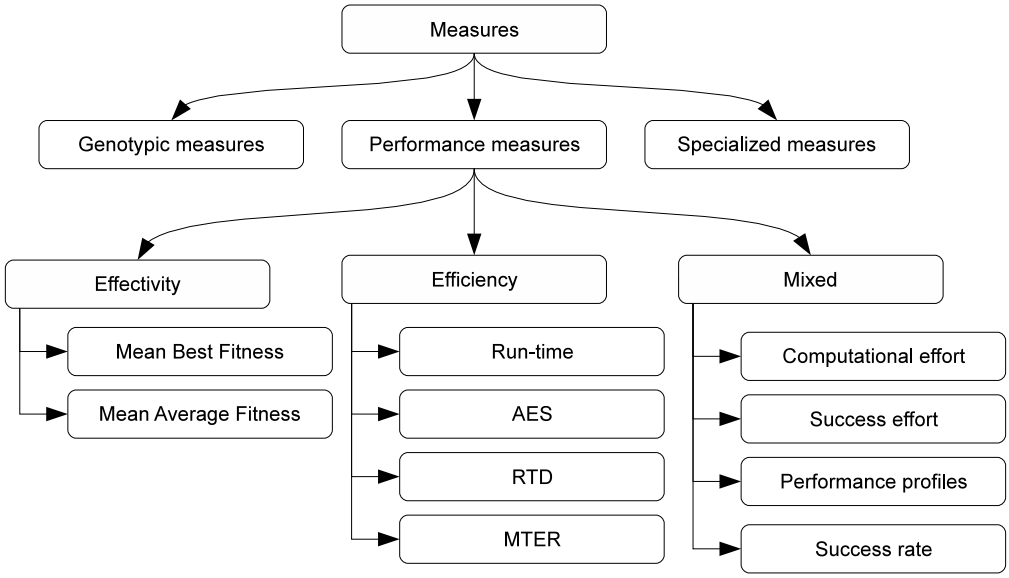


Figure 2.5: Classification of measures in EC with three categories: Genotypic measures, performance measures and specialized measures.

We first mention the work of Rardin and Barr [201, 11], both identify two types of performance measures: solution quality and computation time. They implicitly assume that the run-time can only be measured in time, but for several reasons, measuring run-time in time units generates several troubles [11]. Another term often used to describe the amount of resources needed to achieve a solution is computational effort, which should not be understood as the measure introduced by Koza in [136], which is the keystone of this dissertation, was introduced in the first chapter, and is discussed in detail in chapter 6. Generally speaking, computational effort is understood in this context in a wider sense. Computational effort refers to a set of measures that estimate the amount of resources needed to achieve the solution. To avoid ambiguity, along this section we will use computational effort in a broad sense, and to refer the measure proposed by Koza we will use the term Koza's computational effort.

Bartz-Beielstein, Barr and other authors use the term robustness, however, its meaning strongly changes in function of the context and the author. Bartz-Beielstein uses robustness as a synonym of effectivity [25], as it will be defined later. On the contrary, for Smit et al. [217], the term robustness is used to refer the variance of the output of a certain algorithm. Barr and Eiben use robustness to mean the variation of the algorithm performance when a factor (parameters, problem instance or random seed) is modified [78, 11]. Eiben, on the other hand, emphasizes the generality of the term. In a broader sense, the term robustness is generally used in EC literature to mean the capability of an EA to find efficiently solutions to different problem classes, i.e., a robust algorithm is a generalist algorithm, and thus it is able to continue performing well in case of changes in its environment [200].

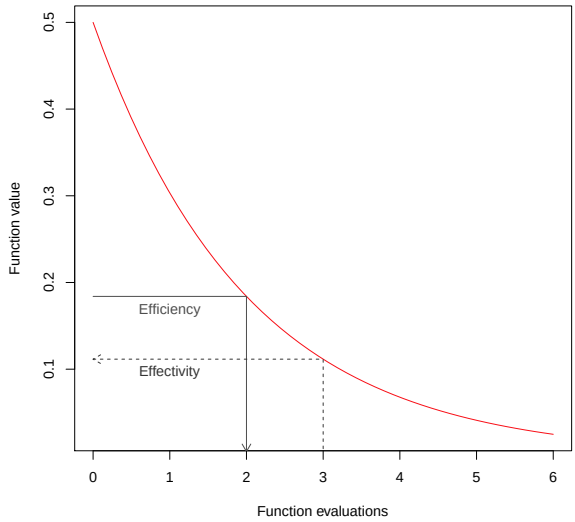


Figure 2.6: Visual representation of effectivity and efficiency measures (source: [25]).

Probably the most common classification scheme of performance measures was proposed by Schwefel in [212], and widely advocated and popularized by Bartz-Beielstein [25] and Eiben [79]. Schwefel's classification scheme of performance measures distinguishes effectivity and efficiency measures. When gathering *effectivity measures*, the amount of resources that this algorithm is allowed to waste is limited, and the measure estimates the quality of the solution that the algorithm is able to find. On the contrary, in *efficiency measures* the solution quality is fixed in advance, and then it estimates how much effort the algorithm needs to find a solution with the given quality. Here, the terms effort and resources should be understood in a broad sense: it might be time, function evaluations, or any other measure able to provide information about how hard is finding a solution. Figure 2.6 represents graphically the difference between effectivity and efficiency. We can identify a third type of performance measure that is not effectivity neither efficiency, but rather a combination of both. These measures are typically a composition of effectivity and efficiency measures, so we will name them *mixed performance measures*. Figure 2.6 represents graphically the difference between effectivity and efficiency .

Effectivity measures are generally related to fitness measurement, we can mention two: Mean Best Fitness and Mean Average Fitness. The *Mean Best Fitness* (MBF) is calculated as the average of the fittest individual in different runs. In other words, the MBF reflects the fit of the best solution that is found. This measure is particularly important in design domains, where this type of extreme individuals and high variance are desirable [79]. Similarly, the *Mean Average Fitness* (MAF) is the average fitness calculated for several runs. Probably, MAF is one of the most used and well known measures. Normality assumption about the distribution of MAF is generally done, however it is not always normal [160] and Rand [200] proposed to express MAF as a ratio between the individuals and the best individual so far

obtained. A third way to measure the effectivity of an algorithm is the performance profiles, proposed by Barreto in [23], which relates the algorithm performance to a set of problems. Flemming warns about some statistical issues that arise when effectivity measures are normalized and compared [85].

Efficiency measures, instead of being related to fitness, use to be related to the measurement of any resource, typically time. Some common efficiency measures are described in the following.

- **Run-time**. One of the oldest, and most evident efficiency measures, is the run-time of the algorithm, i.e., the time, whatever it was measured, that the algorithm takes. Run-time is also a measure that has generated some debate due to the concerns that emerge when it is analyzed in detail. To be specific, several authors have been concerned about the repeatability of this measure [12, 112]. Run-time, measured in time, is highly dependent on architectural considerations, and also depends on issues related to the operating system, such as multitasking, page faults, and so on. Barr [12] suggests removing these times from the measurement. For this reason run-time measured in one machine is rarely reproducible in other machine [62], even if both had the same hardware and software. Additionally, even in the case that all these problems were solved, we find facts that may bias the results: different programming skills, the amount of time devoted to optimize the code, etc. For these reasons, run-time units reported in the literature moved forward machine-independent measures, such as number of evaluations. Sound measurement of run-time is particularly difficult in parallel algorithms, with several specific difficulties [12]. Despite all these disadvantages, Gent still recommends measuring computation times because it is useful to compare different versions of the same algorithm during its development [92].

- **Average Evaluations to a Solution (AES)**. It is the number of individual evaluations needed, in average, to find an individual of the desired quality [76]. This measure is independent of the architecture since it is only influenced by the algorithm itself. Despite this fact, AES is influenced by the random nature of Metaheuristics, and thus it is a random variable.

- **Run-time distributions (RTD)**. An RTD is defined as the probability of finding a solution from the beginning of the run to a certain time $t$. If time is measured in number of evaluations or any other specific algorithm-dependent property, it is named run-time length distribution (RLD). Perhaps, the most notable feature about RTDs are their ability to completely characterize the statistical properties of the run-time. This feature is used by Hoos to study the run-time of several metaheuristic algorithms to a variety of classical problems such as the CSP and 3SAT [113, 114, 224, 115]. Other authors also used this technique to study Metaheuristics like ACO, among others [53], or to characterize the CSP problem [205]. A detailed description of RTDs, and a review of related literature can be found in chapter 5.

- **Others**. There are alternative efficiency measures, such as MTER [107] or quality-effort relationship [11].

Among the mixed performance measures, probably the most important one is *Success Rate* (SR), or frequency of the optimum as Ridge names it [206]. There is some doubts about the classification of SR, some authors claim that it is an efficiency measure, others claim that it is a effectivity measure, while others argue that SR is both, efficiency and effectivity. The latter is based on the fact that to measure SR, the experimenter sets a budget of computational resources and a solution quality, then he verifies whether the algorithm was able to find or not a solution. On the contrary than other measures, SR is not always defined since it needs a criteria to identify the solution, so, in the absence of that criteria SR cannot be defined [24, 76]. There are also other weakness, some authors tend to interpret SR as a measure of the solution quality, not taking into consideration the existence of evidence that shows a lack of correlation between SR and fitness [156]. Chapter 4 is dedicated to study the statistical properties of SR in EC.

The most important performance measure in the context of this dissertation is *Koza's computational effort*. Koza, in [136, chapter 4] introduced a novel measure to estimate the computational effort required by an algorithm to find a solution. Despite the generality of the definition of Koza's computational effort, that only supposes a generational population-based algorithm, its usage has been restricted to GP, where has been rather popular. The main objectives of the thesis is to study the accuracy of computational effort, which is done in chapter 6. Another example of mixed performance measure is the *success effort*, proposed by Walter et al. in [242]. This measure is a simplification of Koza's computational effort and it is calculated as the ratio between the mean generation when the algorithm finds the solution and the proportion of runs finding a solution.

Performance measures tell a critical part of the story, that is simply how well the algorithm performs. However, it is only a part of the whole story, and thus performance measures alone cannot do all the job, specially when a deeper understanding of the algorithm is required. In these circumstances, genotypic measures are necessary.

### 2.5.3 Genotypic measures

There are several issues concerning the structure of the individuals in the population that, although might not have a direct influence in the quality of the solution that they represent, may provide valuable information about the algorithm internals. Probably the most popular genotypic measure is the individual size. Many EAs use fixed-length individuals, such as canonical GAs or ES, but other branches of EC use variable-size individuals, moreover, there are algorithms whose population of candidate solution are intrinsically variable-size like in tree-based GP. Other algorithms, such as GA, have variations to let the population increase the complexity of the solutions they encode [105, 47, 56].

The reason to use variable-length population is well summarized by Harvey "the most impressive feature of natural evolution if how over aeons organisms have evolved from simple organisms to ever more complex ones with associated increase in genotype lengths" [105]. Variable-length algorithms may self-adapt the size of the individuals to the complexity of the problem, which is an important feature in certain problems. Understanding how the population varies its length is fundamental from a practical perspective to fight against one serious problems in variable-length EAs, *code bloat*, which is an increase of the size of the individuals without a correlation with fitness. Code bloat has been object of a intense research in

GP, and there is a large corpus of theories trying to explain it. Two good reviews of the bloat theories in the context of GP can be found in [215, 160]. Detailed discussions about these can be found in [144, 155, 157, 101, 160, 130].

EC literature uses to describe EAs as the action of two opposing forces, exploration and explotation [77]. In the context of Metaheuristics, sometimes the terms intensificacion and diversification are used instead of explotation and exploration [43]. Explotation refers to the capacity of an EA to find promising areas of the search space, while exploration deals with finding the best solution in a certain area of the search space. It is generally recognized that the success of an EA depends on a adequate balance between both, which is determined by how the population is located in the search space. It can be measured using diversity measures. Several diversity measures have been proposed without a clear winner. An important point about diversity measures that should be underlined is its close relationship to representation, because diversity is measured in the genotypic space. Nonetheless, some authors have noticed a correlation between diversity at genotypic and fenotypic levels [118]. A good review about these metrics in GP can be found in [48].

### 2.5.4 Specialized measures

EC involves a wide range of algorithms and problems, thus not surprisingly there are a set of measures that cannot be used in a general case, but instead they only make sense in relation to specific algorithms and problems. These measures might reflect some specific characteristic of the algorithm or problem or, less likely, for cultural reasons it has not being used outside of a certain context. This is the case, for instance, of Koza's computational effort and hits in GP. In this section we briefly present some examples of specialized measures.

One well known advantage of EAs is their intrinsic parallelism, which eases addressing computationally complex problems in parallel architectures. If the measurement of the time response of sequential algorithms exhibit some non-trivial difficulties, measuring time response of parallel algorithms is more challenging. The same considerations made to run-time measurement might be done to parallel EAs, but new considerations due to the parallelism arise: there is a need to use new measurements [12]. For instance, a measure that quantifies the improvement of running an algorithm in parallel is the speedup, which is defined as the ratio between the time required by a serial implementation of the algorithm and the time required by the same algorithm when is run on $p$ processors. In this way, the speedup measures the effect of using several processors in comparison with using an alone processor. Other measures of parallel algorithms can be found in [12].

All the EAs discussed up to this point have in common one characteristic, selection is done attending only to one criteria. Even tough the fitness function might use several criteria to evaluate an individual, selection is performed using its output, which is a single value. It turns out that in nature, survival of individuals is given by a series of factors, such as how fast a prey can run, how it mimics its environment and so on. The same can be said in many problems addressed by EAs, where evaluation of individuals might depend on several objectives and each one has to be maximized or minimized. This behaviour where the goal is given by a set of objectives instead of a single one, is the inspiration of multiobjective algorithms (MOEAs) [254]. In multiobjective algorithms the fit of an individual is not given by a single value, but by a vector whose elements describe the fit of the individual in each objetive under

consideration. Many performance measures used by single-objective algorithms are not suitable for MOEAs. In addition, since the fit of an individual is given by a vector, comparison between algorithms is more complex, and new measures are needed [257, 88, 258].

Another context where many performance measures so far discussed are no longer suitable is dynamic optimization [61] (or nonstationaty function optimization [79]). Previous discussions assumed that the problem and its context were stationary, and therefore they remained without changes over the course of the run. However, in many applications this assumption is not valid, the fitness function, or the constrains, might change. Some survey papers about this topic can be found in [120, 176]. Measuring performance in this type of scenarios is a challenging problem. The main problem is no longer finding an optimum in the search space, but instead is finding a sequence of optimum values over time [79], and thus algorithm performance depends on more factors than in static optimization. Wiecker in [243] analyzed several measures of EAs in dynamic environments, identifying three characteristics that describe the particularities of measures in dynamic optimization. The first one is *accuracy*, which is the ability of the algorithm to find candidate solutions close to the the optimum. Closely related to accuracy is the second characteristic, *stability*, which relates how the accuracy changes when the environment is modified. Finally, the *reactivity* is the ability of the algorithm to react quickly to changes in the environment.

## 2.6 Conclusions

There are several reasons why experimental research is needed in EC. The algorithmic simplicity of EAs contrasts to the notable difficulty of their analytical analysis, that make theoretical results scarce, and rarely useful in practice. So, experimental methods have been widely used in EC research. Despite the importance of experimental methods, there is a lack of research on methodological issues. Much literature has been devoted to describe how to perform experimental research, and how to design sound experiments. However, much of these literature is in form of tutorials or lists of tips, and unfortunately there is a lack of attempts to systematize it. It is interesting to note that most of the papers published around this topic come from Metaheuristics, only recently authors with an EC background have begun to publish on this topic.

In an attempt to provide a systematic approximation to experimentation in EC, we have proposed a framework that eases description of experiments. The proposed framework identifies three intrinsic components of an experiment: algorithm, parameter setting and problem, and an extrinsic component, measures. A solid experimental design requires a rational choice of each one of these components, that should be done according to the research question that motivates the experiment. In order to be able to observe the experiment, we also need measures. They are used to collect data, which is the base of any empirical study. We distinguish three types of measures: genotypic measures, special purpose measures and finally performance measures. Performance measures can be divided into three categories. The first one provides information about the quality of the solution that the algorithm is able to find. The second one informs about the cost of finding a solution of a certain quality. The third category of performance measures mixes the two previous categories, providing an aggregate value. One of the best known mixed performance measures in the context of GP is Koza's

computational effort.

Computational effort, as defined by John Koza, is the minimum number of individuals that have to be processed to achieve a solution of the desired quality with a given probability. This is a mixed measure that includes the population size and the success probability of the algorithm. Despite its popularity in GP, several researchers have shown some concerns about this measure, mostly in informal contexts, but there are a lack of documental evidence supporting these claims. In order to gather data supporting or rejecting these concerns, we first need a deeper knowledge about the statistical properties of the success rate, which is a basic component in Koza's computational effort. In particular, we need a characterization of the error associated to the estimation of the success probability. The statistical properties of that estimation, when the time is fixed, is investigated in chapter 4. However, in the next chapter we will introduce the early work that generated the main research question.

# Chapter 3

# Evolutionary Computation from an applied perspective

*The alchemist in their search for gold*
*discovered many other things of greater value*
*Arthur Schopenhauer*

This chapter takes an applied perspective and tries to develop methods for real applications. Several domains are analyzed, paying more attention to the agent-based information extraction and integration platform named Searchy, modifying it to extract data automatically using evolved wrappers. The research summarized here is the first one to be performed in the context of this PhD thesis, the goal was to use Evolutionary Computation (EC) to solve some real world applications. The value of this chapter is double, firstly, the research performed to develop the chapter provided the necessary experience using EC, and secondly, it helped to find the main research question that is addressed in this dissertation.

We have worked in several problems: a logistic application to optimize the drivers routes [194, 195], generate routes inside a building using RFID [196], language induction [98] and data extraction from the Web [50]. In this chapter we mainly focus on data extraction since it motivated the rest of the PhD thesis.

In order to automatically extract data from the Web, we have used a platform named Searchy. Through the use of a set of wrappers, it integrates information from arbitrary sources and semantically translates them according to a mediated scheme. Searchy is actually a domain-independent wrapper container that eases wrapper development, providing, for example, semantic mapping. The extension of Searchy proposed in this chapter introduces an evolutionary wrapper that is able to evolve wrappers using regular expressions. To achieve this, a Genetic Algorithm (GA) is used to learn a regex able to extract a set of positive samples while rejects a set of negative samples.

This chapter is structured as follows. Section 2 provides a general overview of the system architecture. The information retrieval and information integration mechanism used in

Searchy are briefly described in section 3.3. The evolved regex wrapper is presented in section 3.4 followed by a description of the alphabet construction algorithm. Some experiments carried out by the regex wrapper are shown in section 3.6. Section 3.7 describes related work. Finally, some conclussions are summarized.

## 3.1 Introduction

Organisations have to deal with increasing needs of process automation, yielding a grown of the number and size of software applications. As a result there is a fragmentation of information: it is placed in different databases, documents of different formats or applications that hide valuable data. Thus, it originates the creation of information islands within the organisation. This has a negative impact when users need a global view of the information, increasing the complexity and development costs of applications. Usually ad-hoc applications are developed despite their lack of generality and maintenance costs. Information Integration [102] is a research area that addresses the several problems that emerge when dealing with such scenario.

When a bunch of organizations are involved in an integration process, the problems associated with the integration are increased. Some traditional integration problems, such as information heterogeneity, are amplified and new problems such as the lack of centralized control over the information systems arise. One of the most interesting problems in such context is how to ensure administrative autonomy, i.e., limit as much as possible the constrains that the integration might impose to data sources. We have developed a data integration solution called Searchy with the intention of addressing those constrains.

Searchy [22] is a distributed mediator system that provides a virtual unified view of heterogeneous sources. It receives a query and maps it into one or more local queries, then translates the responses from the local schema to a mediated one defined by an ontology and integrates them. It separates the integration issues from the data extraction mechanism, and thus it can be seen as a wrapper container that eases wrapper development. It is based on Web Standards like RDF (Resource Description Framework) or OWL (Web Ontology Language). Thanks to that, Searchy can be easily integrated in other platforms and systems based on the Semantic Web or SOA (Service Oriented Architecture) and used for other tasks, such as parameter tuning [17, 13].

Experience using Searchy in production environments has shown that some issues need to be improved. One of the most successful wrappers in Searchy was the regex wrapper, a wrapper that extracts data from unstructured documents using a regular expression (or simply *regex*). Regex is a powerful tool able to extract strings that match a given pattern. Two problems were found related to wrapper-based regex utilization: the need of an engineer (or a specialized user, which we usually denoted as wrapper engineer) with specific skills in regex programming, and the lack of automatic way to handle errors in the extraction process. These problems lead us to adapt the Searchy architecture to support evolved wrappers. That is, wrappers based on regex that have been previously generated using Genetic Algorithms (GAs). This wrapper uses supervised learning to generate a regex able to automatically extract records from a set of positive and negative samples.

## 3.2 Searchy architecture

Many Searchy properties are a direct consequence of two design decisions: the MAS approach [3] and the Web standards compliance. Using MAS gives Searchy a distributed and decentralized nature well suited for the integration scenario described in the introduction. Web Services are used by Searchy agents as an interface to access their functionalities, meanwhile the Semantic Web standards are used to provide an information model for semantic and structural integration [238]. From an architectural point of view agents were designed to maximize modularity decoupling integration from extraction issues, easing the implementation of extraction algorithms.

In our architecture, each agent has four components, as can be seen in Figure 3.1. Some of the key properties of Searchy are directly derived from this architecture. These elements are the communication layer, the core, the wrappers and the information source. The next paragraphs describe these components related to the FIPA Agent Management Reference Model.

**Communication layer** It provides features related to the communications such as SOAP message processing, access control and message transport. The Communication layer is equivalent to the Message Transport System (MTS) in the FIPA model.

**Core** It contains the basic skills used by all the agents, including configuration management, mapping facilities or agent identification. Any feature shared by all the agents is contained in the core. It presents some of the features defined by FIPA for the Agent Management System (AMS), however they are not equivalent. AMS are supposed to control the access of the agents to the Agent Platform (AP) and their life cycle. Meanwhile the agent core supports the operation of the wrappers.

**Wrapper** A wrapper is the interface between the core agent and a data source, extracting information from the mediated data source. Wrappers are a key point in order to achieve generality and extensibility. Agents in the FIPA model have some similarities with Searchy wrappers from an architectural point of view. An AP in the FIPA model may contain several agents meanwhile each agent in Searchy may contain several wrappers. Both of them are containers for some software asset, agents in case of FIPA or wrappers in case of Searchy.

**Data source** It is where information that is the object of the integration process is stored. Almost any digital information source might be used as data source. Due to the nature of Searchy, data sources are usually some kind of information system such as a web server or an index. However any source of digital information is a potential Searchy data source. There is no equivalent in the FIPA model to data sources.

Figure 3.1 shows the architecture of a Searchy agent with its four components. Agent interfaces are published thought the HTTP server, one of the subsystems of the communication layer. It receives the HTTP request that has been sent by the Searchy client and extracts the SOAP message. In order to provide a first layer of security, the HTTP subsystem filters the request using the Access Control Module. This module is an IP based filter that enables basic access control. The HTTP server has responsibilities with the SOAP messages transport,

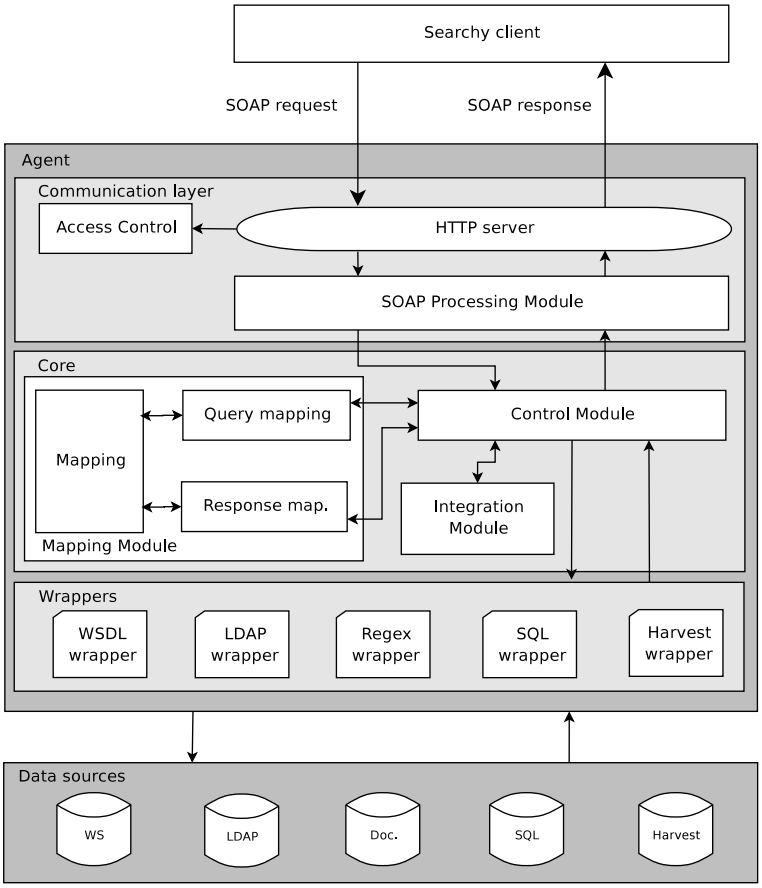


Figure 3.1: Searchy platform architecture, the four components of the architecture are identified in the figure: Communication layer, core, wrappers and data sources.

but the processing of these messages is done by their own module, the SOAP Processing Module. It processes SOAP messages and then transfers operation to the Control Module, or returns an error message. Once the message has been successfully processed, the Control Module starts.

The Control Module sets the flow of operations that the different elements involved in the integration must perform, including the wrappers, the Mapping Module, and the Integration Module. The Mapping Module is composed of three subsystems, with different responsibilities in the mapping process. The Query Mapping subsystem performs query rewriting, translating the query from the mediated schema into the local schema, for example, SQL. Meanwhile, the Response Mapping subsystem translates the response from a local schema like SQL, into RDF, following a mediated schema defined by an ontology. Both, Query and Response Mapping subsystems use the Mapping subsystem, that provides common services related to mappings and rule management to the Query and Response Mapping subsystems. The way in which the integration and mapping processes operate is described in section 3.3. Responsibility for Information extraction, as well as communication among the agents, falls

in the wrappers.

In our architecture, the coordination among agents is based on an organizational structuring model with two different discovery mechanisms. In the first mechanism, each agent has a static knowledge about which agents it must query, where it can find them, and how to access them. The result is a static hierarchical structure. It is useful in order to adapt a Searchy deployment to the hierarchy of a organisation, however it cannot take full advantage of a MAS such as parallelism, the reliability of the whole system is reduced and it is difficult to integrate in dynamic environments.

To overcome some of these disadvantages, a second coordination mechanism has been implemented. Using our previous organizational structuring model, relationships among the agents are not stored within the agents, but externally in a WSDL document that can be fetched by any agent from a HTTP or FTP server. This agent discovery mechanism is simpler than using an UDDI (Universal Description, Discovery, and Integration) directory or a Directory Facilitator (DF) in a FIPA platform. Agents are accessed as another data source, and thus it is done by a set of wrappers responsible of the discovery and communication between Searchy agents: the Searchy and WSDL wrappers. These wrappers implement the coordination mechanism in Searchy, however wrappers' main purpose is to extract data from data sources.

At the present moment, Searchy includes four ordinary wrappers: SQL, LDAP, Harvest and regex. By means of SQL and LDAP wrappers, structured data in databases and LDAP directories may be accessed. Using the Harvest wrapper, Searchy can integrate resources available in an intranet like HTML, LaTeX, Word, PDF documents and other formats. The support of new data sources is done by the development of new wrappers. There is no restriction on the algorithm and data source that the wrapper might implement, it may be a direct access to a database, a data mining algorithm, or data obtained from a sensor. Mapping and integration issues are managed by the agent's core, and thus the wrapper has not to be concerned by these issues. Next section describes how these tasks are performed.

## 3.3 Mapping and integration in Searchy

Integrating information means dealing with heterogeneity in several dimensions [238]. Technical heterogeneity can be overcame by selecting the proper implementation technology. In our work, it has been done using Web Services (WS) as an interface to access to the service. Addressing information heterogeneity requires the definition of a global information model, the mediated schema, among all the entities involved in the integration process, as well as a mapping mechanism to perform a mapping between the different local information models and the global information model. Defining this model is a critical challenge in an information integration system.

Searchy uses semantic technologies standardized by the WWW -RDF, RDFS and OWL- to represent the integrated information. RDF is basically an abstract data model that can be represented using several syntaxes. Searchy uses RDF, serialized with XML, to represent information. This combination of RDF and XML grants interoperability in a structural level. Semantic integration requires an agreement about the meaning of the information to deal with semantic heterogeneity. This agreement is performed by using shared ontologies expressed

in RDFS or OWL. Then, there must be an explicit agreement among all the actors involved in a Searchy deployment to establish at least one global ontology. A set of mapping rules are needed in order to map entities according to a local schema into the global schema. Rules are used to map queries to a local schema, and responses to the mediated schema.

Query format is a tuple <attribute, query> of strings, the Query Mapping subsystem rewrites the query to obtain a valid query for the local data source. The first element in the tuple is an URI that represents the concept to which the query is referred, while the query is a string with the content of the concept that is being queried. The query model is simple but enough to fulfill the requirements of the application. The translation of the query to the local schema is performed using the Mapping Module (see Figure 3.1). Mappings are done by means of a string substitution mechanism, very similar to the traditional *printf()* function in C. This mechanism is enough to satisfy the needs in almost all cases. Once a query has been translated, the response of the local information source must be extracted, mapped to a shared ontology and integrated, respectively, by the Response Mapping and Integration subsystems.

Response mappings are done in two stages:

1. The response is mapped semantically, conforming to a shared ontology. It is done using the same mechanism than the Query Mapping subsystem. A critical aspect is to provide a URI identifier for each resource, just like RDF requires to identify any resource. There is no unified way to do this task: each type of wrapper and user policy define a different way to name resources.

2. Every response of each wrapper is integrated in the Integration Module. Integration is based on the URI of the resource, returned by the wrappers. When two wrappers return two resources identified by the same URI, the agent interprets that they are referred to the same object, and thus they are merged.

Figure 3.2 shows a simple example of an integration process within Searchy. There are two data sources: a relational database, and an LDAP directory service. In a first stage, the wrappers retrieve the information from the local data source, and this is mapped into a RDF model. The mapping is done by using the terms defined by an ontology and according to some rules given by the system administrator. The ontologies used within the integration process must be shared among all agents. In general, a one to one correspondence between a data field and an ontology term will be defined. Several local fields or fixed texts may compose one value in RDF, this feature aids the administrator to define more accurate mappings. The mapping rules defined in the example shown in Figure 3.2 for the database wrapper are depicted in Example 1.

**Example 1** Query mapping rules example

```
rdf:about IS "http://www.example.org/" + name
dc:title IS name + " " + surname
foaf:family_name IS surname
```

The first rule defines that the RDF attribute *rdf:about* is built with the concatenation of the string ”http://www.example.org/” and the attribute Person as it is defined in the local

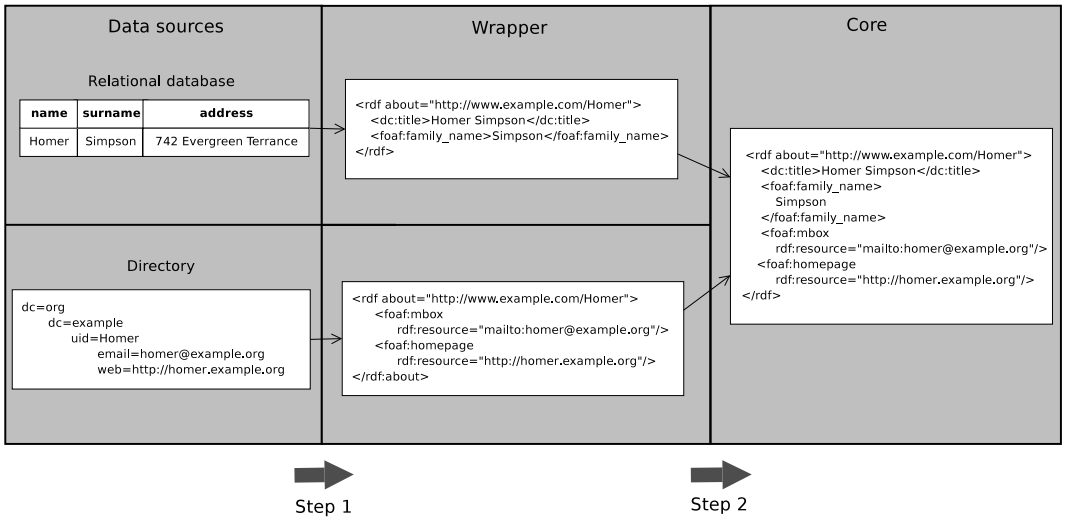


Figure 3.2: Example of the integration process in Searchy, with two data sources, one relational database and a directory.

schema. The rest of rules are defined in a similar way. Meanwhile, the mapping rules for the directory wrapper can be seen in Example 2.

**Example 2** Response mapping rules example

```
rdf:about IS "http://www.example.org/" + uid
rdf:type IS foaf:Person
foaf:mbox IS email
foaf:homepage IS web
```

The wrappers in the example use two vocabularies: Dublin Core and FOAF. Each object retrieved from the data source must be identified by an URI, that in this case is built using local data with a fixed text. The second stage integrates the entities returned by the wrappers. The agent core identifies the two objects as the same object by comparing their URI and merges the attributes, providing a RDF object with attributes retrieved from two different sources.

Mapping and Integration Modules decouple data integration and mapping from the extraction, and thus it is possible to develop wrappers in Searchy without any concern about these issues. Next section shows an example of how a complex wrapper may be developed using the infrastructure provided by Searchy.

The original architecture of Searchy [14] provided an easy to use extraction and integration platform. However, it required human supervision in some parts of the process. One of the most useful wrappers supported by Searchy is the regex wrapper, which is able to extract data from unstructured documents. One problem associated with this wrapper is the need of a wrapper engineer skilled in regex programming. Another problem is error detection, that is, detect when the wrapper is not correctly extracting data and solve it.

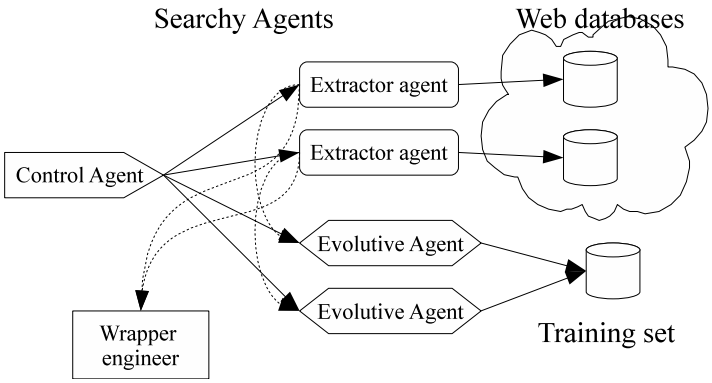


Figure 3.3: Example of a Searchy deployment with extractor, evolutive and control agents.

It lead us to extend the original Searchy regex wrapper able to extract data using a regex created by the wrapper engineer with an evolved regex agent able to generate a regex from a set of positive and negative examples using a GA. Figure 3.3 depicts the extended architecture, where the original architecture is extended with control and evolutive agents. The MAS contains three kind of agents: control, extractor and evolutive agents. The three types of agents share the same agent architecture depicted in Figure 3.1, they differ from an architectural point of view in the wrappers they use. Figure 3.3 uses solid lines to represent the iteration among the agents and resources with the exception of iterations that involve regex, which is represented with dotted lines.

There must be one control agent that receives queries from the user and forwards it to the extractors, which are agents with a regex wrapper. Regex wrappers in the original Searchy architecture obtained the regex from the wrapper engineer, who manually generated the regex. When the wrapper detected a failure in the data extraction, i.e., when it was unable to extract data from a source, the wrapper notified it to the wrapper engineer who had to identify the problem and in case the regex was incorrectly constructed, generates a new one.

The new architecture aims to automate this approach, using an evolutive agent that fulfills some roles of the wrapper engineer. Extraction agents obtain the regex from the evolutive agents at start-up time, but also when they identify an extraction error. In this case, instead of requesting a new regex to the wrapper engineer, it would request it to the evolutive agent. When an evolutive agent is required to generate a new regex, it executes a GA as described in the next section.

## 3.4 Wrapper based on evolved regular expressions

The implementation of the evolved regex was done as a Searchy wrapper using the Searchy wrapper API. When an agent with the evolved regex wrapper is run, the wrapper generates a

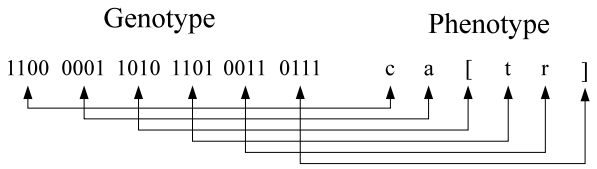


Figure 3.4: Example of chromosome encoding.

valid regex executing the described VLGA with a given training set. Once a suitable regex is generated, the wrapper can begin to extract records from any text file accessible thought HTTP or FTP. It does not have to manage any mapping-related issue since the Mapping Module performs this task.

### 3.4.1 Codification

Any GA has to set a way to codify the solution into a chromosome. The VLGA implemented in the wrapper uses a binary genome divided in several genes of fixed length. Each gene codes a symbol $\sigma$ from an alphabet $\Sigma$ composed by a set of valid regular expressions constructions, as described in section 3.5.

Some words should be dedicated to how genes code regex. The alphabet is not composed by single characters, but by any valid regex, in this way the search space is restricted leading to a easier search. These simple regular expressions are the building blocks of all the evolved regex and cannot be divided, thus, we will call them atomic regex. The position (or *locus*) of a gene determines the position of the atomic regex. Gen in position $i$ is mapped in the chromosome to regex transformation as an atomic regex in the position $i$. Figure 3.4 represents a simple example of how the regex *ca[tr]* could be coded in the GA.

### 3.4.2 Evolution strategy

Genetic operators used in the evolution of regular expressions are the mutation and crossover. Since the codifications rely in a binary representation, the mutation operator is the common inverse operation, while the recombination is performed with a cut and splice crossover. Given two chromosomes, this operator selects a random point in each chromosome and use it to divide it in two parts, then they are interchanged. Obviously, the resulting chromosomes will likely be of different lengths. Selective pressure is introduced by a tournament selection where $n$ individuals are randomly taken from the population and the one that scores the higher fitness is selected for reproduction. An elitist strategy has also been used, where some of the best individuals in the population are transferred, without any modification to the new generation. In this way it is assured that the best genetic information is not lost.

### 3.4.3 Fitness

How the goodness of any solution is measured is a key subject in the construction of a GA. In our case, for each positive example, the proportion of extracted characters is calculated. Then the fitness is calculated subtracting the average proportion of false positives in the negative example set to the average of characters correctly extracted. In this way, the maximum fitness that a chromosome can achieve is one. This happens when the evolved regex has correctly extracted all the elements of positive examples while none of the negative examples has been matched. An individual with a fitness value of one is called *ideal individual*.

From a formal point of view, the fitness function that has been adopted in the wrapper uses a training set composed by a positive and a negative subset of examples. Let $\mathcal{P}$ be the set of positive samples and $\mathcal{Q}$ the set of negative samples, such as $\mathcal{P} = \{p_1, p_2, ..., p_M\}$ and $\mathcal{Q} = \{q_1, q_2, ..., q_N\}$. Both, $\mathcal{P}$ and $\mathcal{Q}$ are subsets of the set of all strings $\mathcal{G}$, and they have no common elements, so $\mathcal{P} \cap \mathcal{Q} = \emptyset$.

Chromosomes are evaluated as follows: Given a chromosome, it is transformed into the corresponding regex $r \in \mathcal{R}$, then tries to match against the elements of $\mathcal{P}$ and $\mathcal{Q}$. The set of strings that $r$ extracts from a string $p$ is given by the function $\varphi(p, r) : (\mathcal{S} \times \mathcal{R}) \longrightarrow \mathbb{R}$ while the number of characters retrieved is represented by $|\varphi(p, r)|$. The percentage of extracted characters of $p_i$ such as $i = 0, ..., M$ is averaged, and finally the fitness is calculated subtracting the average proportion of false positives in the negative example set to the average of characters correctly extracted, as expressed by:

$$\mathfrak{F}(r) = \frac{1}{|P|} \sum_{p_i \in \mathcal{P}} \frac{|\varphi(p_i, r)|}{|p_i|} - \frac{1}{|Q|} \sum_{q_i \in \mathcal{Q}} M_r(q_i) \tag{3.1}$$

where $|p_i|$ is the number of characters of $p_i$, $|P|$ the number of elements of $\mathcal{P}$, $|Q|$ the number of elements of $\mathcal{Q}$ and $M_r(q_i)$ is defined as

$$M_r(q_i) = \begin{cases} 1 & if \quad |\varphi(q_i, r)| > 0 \\ 0 & if \quad |\varphi(q_i, r)| = 0 \end{cases} \tag{3.2}$$

## 3.5 Zipf's law based alphabet construction

### 3.5.1 Preliminary considerations

Section 3.4.1 has shown how a classical binary codification is used to select one symbol $\sigma$ from a predefined set $\Sigma$ of symbols or atomic regex. The construction of $\Sigma$ is a critical task since it determines the search space, its size and its capacity to express a correct solution. Of course, the simplest approach is to manually select the alphabet, however this approach may devaluate the added value of evolved regex: the automatic generation of regex.

We can state that the construction of $\Sigma$ must satisfy three constrains.

1. $\Sigma$ must be *sufficient*, i.e., it must exist at least an element $r \in \Sigma^*$ such as $r$ is an ideal individual. In other words, it must be possible to construct at least one valid solution using the elements of $\Sigma$.

2. $|\Sigma|$ must contain the minimum number of elements able to satisfy the sufficiency constrain. Of course, being able to satisfy this condition is a challenging task with deep theoretical implications. From a practical point of view, this constrain can be reformuled as trying to keep $|\Sigma|$ as small as possible.

3. Symbol selection must be automatic, with minimal number of parameters and human interaction.

### 3.5.2 Alphabet construction algorithm

To reduce the number of elements of $\Sigma$, and keep the search space as small as possible, we aim to identify patterns in the positive samples and use them as building blocks. In order to satisfy the previous constrains we propose the following algorithm. $\Sigma$ is built as the union of $\mathcal{F}$, $\mathcal{D}$ and $\mathcal{T}$, where $\mathcal{F}$, is the set of fixed symbols, $\mathcal{D}$ the set of delimiters and $\mathcal{T}$ the set of tokens.

$$\Sigma = \{\sigma_i\} \backslash \sigma_i \in \mathcal{F} \cup \mathcal{D} \cup \mathcal{T} \tag{3.3}$$

**Algorithm 1** Selection of alphabet tokens.

```
1 .- P := Set of positive examples
2 .- S := Set of candidate delimiters
3 .- D := T := { }
4 .-
5 .- for each p in P
6 .-     for each s in S
7 .-         tokens := split p using s
8 .-         numberTokens := number of tokens
9 .-
10.-         for each token in tokens
11.-             occurrence(token) := occurrence(token) + 1
12.-         endfor
13.-
14.-         if (numberTokens > 0) add s to D
15.-     endfor
16.- endfor
17.-
18.- sort occurrence
19.- add n first elements of occurrence to T
```

$\mathcal{F}$ contains manually created reusable symbols that are meant to be common cross-domain regex, and thus, once they have been defined they can be used to evolve different regex. It should be noticed that $\mathcal{F}$ may contain any valid regex, nevertheless it is supposed to contain generic use regex such as $\backslash d+$ or *[1-9]+*. Since $\mathcal{F}$ is supposed to include common

used complex regex, it contributes to reduce the search space and increase individual fitness by introducing high fitness building blocks.

The sets $\mathcal{D}$ and $\mathcal{T}$ are constructed using a more complex mechanism based on Zipf's Law [256]. It states that occurrences of words in a text are not uniformly distributed, rather only a very limited number of words concentrates a high number of occurrences. This fact can be used to identify patterns in $\mathcal{P}$, and use them to construct a part of $\Sigma$.

Since the tokens do not contain delimiters, the sufficiency constrain cannot be satisfied, so, each delimiter that appear in the examples is included in set $\mathcal{D}$. The overall process is described in Algorithm 1. Of course, $|\Sigma|$ must be equal to the number of elements of the union of $\mathcal{F}$, $\mathcal{D}$ and $\mathcal{T}$, as is expressed in equation (3.4).

$$|\Sigma| = |\mathcal{F} \cup \mathcal{D} \cup \mathcal{T}| \tag{3.4}$$

given that

$$|\mathcal{F} \cap \mathcal{D} \cap \mathcal{T}| = |\mathcal{F} \cap \mathcal{D}| = |\mathcal{F} \cap \mathcal{T}| = |\mathcal{D} \cap \mathcal{T}| = \oslash \tag{3.5}$$

### 3.5.3 Complexity analysis

A better understanding of the algorithm can be achieved by a time complexity analysis. As can be seen in Algorithm 1, there are two main loops (see Algorithm 1, lines 5 and 6) that depend on the number of examples $|P|$, and the number of potential delimiters $|S|$. The complexity of the algorithm is given by these loops and the operations that are performed inside.

Splitting a string $p_i \in \mathcal{P}$ (line 7) is proportional to the length of the string $|p_i|$, so the mean time required to perform this operation is proportional to the mean string length $|\overline{p}|$. Lines 19 to 21 include a loop that is repeated as many times as tokens are in the string. A hash table is accessed inside the loop (line 20), so it makes sense to suppose that its complexity is given by the computation of the key, a string, therefore its time complexity is $n|\overline{p}|$, where $n$ is the number of tokens. Finally sorting $occurrence$ can be performed in $n_{tot}\log(n_{tot})$ where $n_{tot}$ is the number of tokens stored in $occurence$. The rest of operations in the algorithm can be performed in negligible time. We can express these considerations in equation (3.6).

$$t \propto |P| \cdot |S| \cdot [|\overline{p}| + n|\overline{p}|] + n_{tot}\log(n_{tot}) \tag{3.6}$$

Both $n$ and $n_{tot}$ are unknown and we have to estimate them for the average case. A string $p \in P$ of length $|p|$ can contain approximately $\frac{|\overline{p}|}{2}$ tokens. We have supposed there is one delimiter for each token. The maximum number of tokens that can be stored in $occurrences$ are $\frac{|P|\cdot|S|\cdot|\overline{p}|}{2}$. Then

$$n = \frac{|\overline{p}|}{2} \tag{3.7}$$

$$n_{tot} = \frac{|P| \cdot |S| \cdot |\overline{p}|}{2} \tag{3.8}$$

and 3.6 can be expressed as

$$t \propto |P| \cdot |S| \cdot |\overline{p}|[1 + \frac{|\overline{p}|}{2}] + \frac{|P| \cdot |S| \cdot |\overline{p}|}{2} \log(\frac{|P| \cdot |S| \cdot |\overline{p}|}{2}) \tag{3.9}$$

Some terms can be removed

$$t \propto \frac{|P||S||\overline{p}|}{2} \log(\frac{|P| \cdot |S| \cdot |\overline{p}|}{2}) \tag{3.10}$$

Using Big O notation, it yields that the time complexity is given by

$$O(k \log(k)) \tag{3.11}$$

where $k = |P||S||\overline{p}|$ and hence we can conclude that the time complexity is linearithmic.

## 3.6 Evaluation

Two phases have been used in the evaluation, a first phase where the basic behaviour of the GA is analyzed, and a second phase that uses the knowledge acquired along the first phase to measure the extraction capabilities of the evolved regex wrapper. Measures that have been used are the well known precision, recall and F-measure. The sets of experiments described in this section are focused in the extraction of three types of data: URLs, phone numbers and email addresses.

### 3.6.1 Parameter tuning

Some initial experiments were carried out to acquire knowledge about the behaviour of the regex evolution and select the GA parameters to use within the wrapper. Experiments showed that despite the differences between phone, URL and emails, all the case studies have similar behaviors. In this way it is possible to extrapolate the experimental results and thus to use the same GA parameters. Setup experiments showed that best performance is achieved with a mutation probability of 0.003 and a tournament size of 2 individuals. A population composed by 50 individuals is a good trade-off between computational resources and convergence speed. Initial population has been randomly generated with chromosome lengths that range from 4 to 40 bits, and elitism of size one has been applied. Table 3.1 summarizes the parameter values used in the experiments.

### 3.6.2 Regex evolution

Once the main GA parameters have been set, the wrapper can evolve the regex. Experiments have used three datasets to evolve regex able to extract records in the three case studies under scrutiny. Figure 3.5 (left) depicts the *Mean Best Fitness* (MBF) and *Mean Average Fitness* (MAF) of 100 runs. The fitness evolution of the case studies follows a similar path. The best MBF and MAF are achieved by the email regex, while the poorest performance is given by the URL regex, with lower fitness values.

The dynamics of the chromosome length can be observed in Figure 3.5 (right). It is clear that there is a convergence of the chromosome length and thus chromosome bloating does not appear. It can be explained by the lack of non-coding and overlapping regions in the

Table 3.1: Summary of the GA parameters used to evolve regular expressions.

| Parameter | Value |
|---|---|
| Population | 50 |
| Mutation probability | 0.003 |
| Crossover probability | 1 |
| Tournament size | 2 |
| Elitism | 1 |
| Initial chromosome length | 4 - 40 |

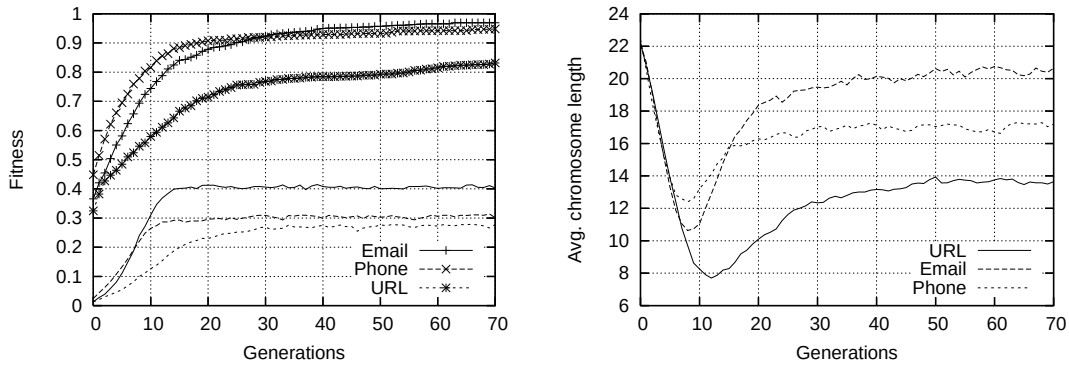


Figure 3.5: Left: Best and average fitness of phone, URL and email regex. Right: Average chromosome length.

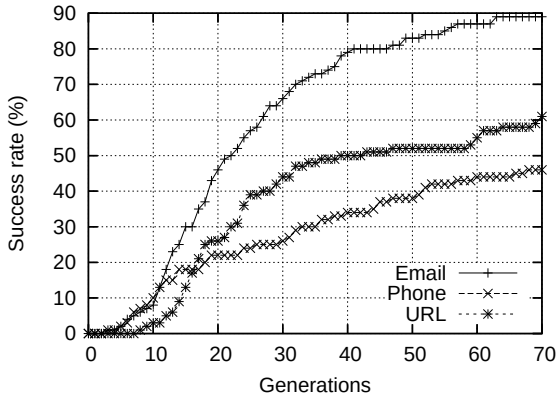


Figure 3.6: Probability of finding an ideal regex able to accept all the positive examples while rejecting the negative ones.

chromosome, i.e, if the chromosome has achieved a maximum it hardly can increase its size without a penalty in its fitness. The longer is the chromosome, the more restrictive is the phenotype and it is closely related to the associated fitness. URL regex has a stronger tendency to local maximum, this fact reflected in Figure 3.5 (right), where lower MBF and MAF are achieved. This fact also explains why URL chromosome length depicted in Figure 3.5 (right) is shorter than phone regex: the local maximum of URL regex tends to generate populations with insufficient chromosome length. Those results are not surprising since URLs follow a far more complex pattern than phone numbers or emails. The same can be affirmed about emails in comparison to phone numbers.

Figure 3.5 (right) shows another interesting behaviour. As the GA begins to run, the average chromosome length is reduced until a point where it begins to increase, then the chromosome length converges into a fixed value. In early generations individuals have not suffered evolution and thus its genetic code has a strong random nature. Individuals with longer genotype have longer phenotypes and thus more restrictive regex that will likely have smaller fitness values. So long chromosomes are discarded at early stages of the evolutive process until the population is composed by individuals representing basic phenotypes, then recombination leads to increased complexity in individuals until they reach a length associated with a local or global maximum.

Some of the facts found previously are confirmed by Figure 3.6, where the success rate (SR) [15] is depicted versus the generation. SR is defined as the probability of finding an ideal individual in a given generation. It should be noted that Figure 3.6 depicts the average success rate of 100 runs of the experiment. It can be seen that email achieves a SR of 91%, phone numbers 60% and URLs 46% by generation 70. These results are consistent with those in Figure 3.5 (right), and show that the hardest study cases are URLs, phone numbers, and emails, in that order. Here the term ”hard” should not be understood in a strict absolute way since the hardness of the search space is influenced from several factors, such as the training set, the selection of negative samples, or the chosen alphabet.

Table 3.2: Extraction capacity of the evolved regex. The table shows the F-measure (F), precision (P) and recall (R) achieved in the three datasets (phone, URL and email addresses).

| | | | | **Phone regex** | | | **URL regex** | | | **Email regex** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Ph. | URL | Email | **F** | **P** | **R** | **F** | **P** | **R** | **F** | **P** | **R** |
| Set 1 | 99 | 0 | 0 | 1 | 1 | 1 | - | - | - | - | - | - |
| Set 2 | 0 | 51 | 0 | - | - | - | 0.24 | 0.14 | 0.84 | - | - | - |
| Set 3 | 0 | 0 | 862 | - | - | - | - | - | - | 0.79 | 0.51 | 0.62 |
| Set 4 | 20 | 77 | 0 | 1 | 1 | 1 | 0.27 | 0.16 | 1 | - | - | - |
| Set 5 | 37 | 686 | 0 | 1 | 1 | 1 | 0.20 | 0.11 | 0.97 | - | - | - |
| Set 6 | 24 | 241 | 0 | 1 | 1 | 1 | 0.02 | 0.01 | 0.37 | - | - | - |
| Set 7 | 83 | 0 | 88 | 0.92 | 1 | 0.96 | - | - | - | 0.92 | 1 | 0.96 |
| Set 8 | 0 | 51 | 0 | - | - | - | 0.63 | 0.47 | 0.96 | - | - | - |
| **Avg.** | - | - | - | 0.98 | 1 | 0.99 | 0.27 | 0.18 | 0.83 | 0.85 | 0.79 | 0.79 |

### 3.6.3 Data extraction

Three regex with an ideal fitness of one have been selected by the wrapper and its extraction capabilities have been evaluated by means of the precision, recall and F-measure. The experiments used a dataset composed by eight sets of documents from different origins containing URLs, emails and/or phone numbers. Table 3.2 shows basic information about the datasets and their average records and Table 3.3 contains some evolved regex with their fitness value. Sets one, two and three are composed by examples extracted from the training set. The rest of the sets are web pages retrieved from the Web classified by their contents. An extracted string has been evaluated as correctly extracted if and only if it matches exactly the records, otherwise it has been computed as a false positive.

The results, as can be seen in Table 3.2, are quite satisfactory for phone numbers and testing sets, but measures get worse for real raw documents, specially the ones containing URL records. Phone regex has a perfect extraction with a F-measure value close to 1. The training set used to evolve regex contains phone numbers in a simple format $(000)000-0000$, the same that can be found in the testing set, the reduction of recall in set 7 is due to the presence of phone extensions that are not extracted.

On the contrary, measures achieved for URL extraction from raw documents are much lower. It can be explained looking at the regex used in the extraction, *http://\w+\.\w+\.com*. Documents used in the test contain many URLs with paths, so the regex is able to partially extract them, increasing the count of false positives. The result is a poor precision. An explanation of the poor recall measures in URLs extraction is found in the fact that the evolved regex only is able to extract URL whose first level domain is *.com*, so its recall in documents with a high presence of first level domains of any other form is worse.

Finally, email regex achieves an average F-measure of $0.85$. Some of the factors that limits the URL regex extraction capabilities are also limiting email regex. However in this case the effects are not so severe for a number of reasons, for instance the lower percentage of addresses with more than two levels.

Table 3.3: Some examples of evolved regular expressions with their fitness values.

| Evolved regex (Phone) | Fitness |
|---|---|
| \w+ | 0 |
| \(\d+\) | 0.33 |
| \(\d+\)\d+ | 0.58 |
| \(\d+\)\d+-\d+ | 1 |
| **Evolved regex (URL)** | **Fitness** |
| http://-http://http:// | 0 |
| /\w+\. | 0.55 |
| http://\w+\.\w+\ | 0.8 |
| http://\w+\.\w+\.com | 1 |
| **Evolved regex (Email)** | **Fitness** |
| \w+\. | 0.31 |
| \w+\.\w+ | 0.49 |
| \w+@\w+\.\com | 1 |

## 3.7 Other approaches to distributed Information Integration

The use of ontologies [100] has attracted the attention of the data integration community over the last years. It has provided a tool to define mediated schemas focused on knowledge sharing and interoperability, in contrast with traditional database centric schemas, whose goal is to query single databases [232]. The adoption of ontologies has lead to reuse results achieved by two communities such as the database and the AI communities to solve similar problems like schema mapping or entity resolution. A deep discussion about the role of ontologies in data integration can be found in [183].

We can define a collection of semantic solutions based on ontology technologies prior to the development of the SW. An introduction to this group of solutions can be found in [238]. We can highlight classical literature examples such as InfoSleuth [181] or SIMS [132]. From these systems, we have to single out InfoSleuth, a solution that uses a MAS.

Semantic integration tools in the last years have adopted WS standards and technologies. One of the first ones can be found in [234]. Vdovjak proposes a semantic mediator for querying heterogeneous information sources, but limited to XML documents; furthermore, this solution relies on a wrapper layer that translates the local entities into XML and only then the RDF is generated. A step forward is achieved by Michalowski with Building Finder [168], a domain specific mediator system aimed at retrieving and integrating information about streets and buildings from heterogeneous sources, presented to the user within satellite images. [253] describes an information integration tool that covers all the phases of integration, such as assisted mapping definition and query rewrite.

Another newcomer into the IT toolbox is the Web Services technology. WS provide a means to access services in a loose coupling way. Despite WS and the SW face different

Table 3.4: Comparison of semantic information integration tools.

| Platform | Agent support | Semantic Web | Web Services | Interdomain support |
|---|---|---|---|---|
| InfoSleuth | Yes | No | No | Yes |
| SIMS | Yes | No | No | Yes |
| Building Finder | No | Yes | No | No |
| SODIA | No | Yes | Yes | Yes |
| Knowledge Sifter | Yes | Yes | Yes | Limited |
| Searchy | Yes | Yes | Yes | Yes |

problems -one models and represents knowledge while the other one is concerned with service provision-, they are related by means of semantic descriptions of WS throught Semantic Web Services. In this way WS are enhanced with semantic descriptions, enabling dynamic service composition and data integration [68].

A semantic integration solution based on SOA is SODIA (Service-Oriented Data Integration Architecture) [255]. It supports some integration approaches such as federated searches and datawarehouses. By using a SOA approach, SODIA has many of the benefits of using an agent technology. However, this is a process centric solution and has limited semantic support. The most aligned solution to the one described in this chapter is Knowledge Sifter [129]. It consists of an agent based approach that uses OWL to describe ontologies and WS as interface to the agents' services. Despite the lack of semantic support, WS integration or distributed nature, we have to mention the system proposed by [211], able to automate the full integration process by creating the mediated schema and schema mapping on-the-fly. Another interesting integration suite related to bioinformatics domain that could be mentioned is INDUS [51].

Table 3.4 compares some representative federated ontology-driven search solutions. The scope of table 3.4 is limited, however some relevant facts are shown. It depicts whether the integration system is supported by agents, it uses any WS or SW technology as well as the degree of specialization of the tool.

## 3.8 Conclusions

We have described a semantically enabled extraction and integration agent-platform named Searchy. This platform basically works as a wrapper container that can be extended using almost any extraction algorithm. Using its capabilities, a new evolutive wrapper based on GAs was introduced. This wrapper, using a set of positive and negative examples tries to generate a regex able to accept the positive examples while rejecting the negative ones. Then, the wrapper is able to extract information using the regex and integrate it. Perhaps the most relevant contribution of the chapter is an algorithm based on Zipf's law used to build an alphabet of symbols that are used in the regex.

However, the results of this chapter should be carefully interpreted. There are some concerns about the experimental design that should be considered. Firstly, there is a strong

dependence between the capabilities of the algorithm and the training dataset that feeds it. In order to make fair experimentation, the effect of the dataset should be taken into account. For this reason, a common practice in Machine Learning is the use of cross-validation, but is has not been applied in the work reported in this chapter, and therefore we cannot exclude the possibility of biased results by the dataset composition.

There are also more substantial concerns. This chapter had a strong engineering flavor since it deals with the development of new methods, and it naturally leads to an "algorithm race" research [112], where the research question deals with which algorithm has better performance. But this question is naturally evil: there are serious issues about the posibility to draw a scientifically solid answer to that question. There are too many factors to take into account, such as datasets or parameter settings, to make a fair comparison. Even in the case that the experimental design was solid, the conclusions that would be obtained could not be generalized.

In addition to these concerns, there are also some pitfals. For instance, Figure 3.6 depicted the success probability of several GAs as a way to compare algorithms. However, this is not a sound comparison method. The figure shows the central tendency of the success probability, but does not characterize the variability of the data. Only with the information provided by the figure, we cannot know if the result if due to the randomness or, on the contrary, if reflects well the reality. For this reasons, Figure 3.6 should be used with care in order to support any claim of that nature. More robust statistical methods, like the ones that are described in chapter 4, are needed.

From the perspective of the algorithm, we have some additional concerns. Despite the success of this platform as wrapper container, several issues emerge from the use of EC to evolve regex. Perhaps the most important has a very basic nature: there are a whole set of domain specific algorithms with good performance able to solve this problem. These algorithms exploit the underlying nature of regular expressions, which are DFAs. Due to the No-Free Lunch theorem, it is difficult for a non-specialist algorithm such as an EA to outperform good specialist algorithms, such as ESDM [142, 141, 153, 57].

In particular, GAs are not well suited for this task. The linear representation used in GAs does not map naturally to regex. This observation motivated us to move forward to tree-based Genetic Programming, which has a representation closer to regex. In order to compare the standard algorithm and some variations that we introduced, we begun using Koza's computational effort, and we observed a high variability in the results. Given this variability, we had difficulties to find differences in the algorithms that we were analyzing. Intrigued by this fact, we begun to study what was happening, finding that there were several problems with the measure. In this way, the main research question of the dissertation was stated. The next chapter begins with the study of Koza's computational effort, focusing on one of its fundamental components: the estimation of a probability.

# Chapter 4

# Estimation of the success rate in Evolutionary Computation

*Oh, people can come up with statistics to prove anything.*
*14% of people know that*
*Homer J. Simpson*

In this chapter we aim to characterize the statistical properties of the static estimation of the success probability, which we name *success rate* (SR). Therefore, SR is the success probability when the algorithm is run for a certain fixed time. In particular, and without loss of generality, we consider that SR is the success probability at the end of the run of an algorithm, when the finish condition is given by a time limit. The characterization of the SR is interesting by its own, given that this is a measure widely used in Evolutionary Computation (EC). A better knowledge about the estimation of SR would provide a basis to introduce more robust statistical methods. In addition, this characterization is used by futher chapters of this PhD thesis in order to develop a time-dependent model of success probability (chapter 5), and characterize the error associated to the estimation of Koza's computational effort (chapter 6).

Along the chapter we provide theoretical and empirical evidences strongly suggesting that the number of success runs in an Evolutionary Algorithm (EA) (and therefore the SR) can be modeled using a binomial distribution. Binomiality of SR implies that all the statistical tools available for binomial distributions can also be used with SR. We review the statistical literature about one of these tools, binomial *confidence intervals* (CIs) and characterize the quality of several methods in the context of EA. In addition, due to its practical interest, we provide a brief discussion of a method that determines the number of runs that an EA should be run to generate CIs with a given quality.

The chapter is structured as follows. Firstly, we motivate the importance of SR in EC, then we introduce the basic problem of estimating a probability. In section 4.3, we study the statistical distribution that models SR, and we continue with a description of several meth-

ods described by the literature to calculate binomial CIs. Section 4.5 studies the performance of several CIs methods in relation to the number of samples of the experiment and SR. In section 4.6, we compare binomial CIs applied to some classical GP problems with CIs applied to theoretical binomial distribution. Section 4.7 briefly describes a method to estimate the number of runs needed to build CIs of a given quality. We finish the chapter with some conclusions.

## 4.1 The role of success rate

Several measures have been used in EC research [23]. The selection of one measure instead of another one depends on the object of study, the algorithm and the goals of the experiment designer [61, 257]. However, there are some common measures that are heavily used such as mean best fitness or mean average fitness [79]. One of the most common ones is the SR. Due to the stochastic nature of EAs, when an EA is run it might, or might not, reach a solution that solves the problem it was designed for. SR is defined as the probability of an EA to find such a solution, which is determined by imposing a success predicate, for instance, when an individual achieves a certain quality. In other words, it is not possible to use SR if there is not a criteria to identify an enough good solution [76, 24].

SR should be used with caution. As Luke and Panait [159] noticed, fitness might not be correlated with SR, and consequently it should not be used as a measure of the population quality. Nonetheless, finding literature that reports SR as a quality measure of the population is not too hard. In any case, SR provides an insight to the capabilities of the algorithm to find a solution. Some times SR is not the measure of interest, but rather it is part of a complex measure such as the *computational effort* [136] in Genetic Programming (GP). In this case the accuracy of the complex measure depends on the quality of the estimation of SR.

One characteristic of SR as has been defined above is that it is defined as a scalar, but the value of the scalar cannot be known in the general case. It has to be estimated. Angeline [6] was the first person to observe this fact when working with computational effort, and suggested that a measure about a stochastic process should take into account its random nature. The same can be said of SR, a single point is not enough to characterize the stochastic nature of this metric, and some additional information about its statistical properties should also be reported, for instance, CIs. A number of issues arise when the stochastic nature of SR is analyzed in detail.

## 4.2 Issues about the estimation of a probability

From the perspective of SR an EA experiment is just a Bernoulli trial: an EA run is just an experiment whose outcome is a binary random variable $X$ that can take two values, let's call them "success" or "failure". SR is defined as the probability $P(X = \text{“}success''\text{)} = p$, and is described by the Bernoulli distribution. The most common case in EC is that $p$ is unknown, which is precisely the parameter we want to estimate. The procedure to do it is well known, the EA is repeated $n$ times yielding a number $k$ of successes and $n - k$ failures, then a probability $p$ is computed as $p = \frac{k}{n}$. Actually, we have described a Bernoulli process, a sequence $X_i, i = 1, 2, ..., n$ of random variables that are the outcome of a sequence of

Table 4.1: Simulation of the estimation of the probability of getting heads when 1000 coins are tossed.

| Experiment | Successes | $\hat{p}_i$ |
|---|---|---|
| 1 | 483 | 0.483 |
| 2 | 531 | 0.531 |
| 3 | 594 | 0.594 |
| 4 | 521 | 0.521 |
| 5 | 513 | 0.513 |
| Total | 2642 | $\frac{2642}{5000} = 0.5284$ |

independent Bernoulli trials, and therefore they are described by a Bernoulli distribution. Despite its simplicity, a number of trivial and non-trivial issues arise when this experiment is analyzed in more detail.

Consider the next naíve experiment. We aim to empirically measure the probability of obtaining head when a coin is tossed. Of course, if the coin is equilibrated and the experiment is well implemented, that probability is $1/2$. But we want to study it empirically, so, we try the experiment $3$ times and count the number of successes. In this case there are only four possible outcomes, $k \in \{0, 1, 2, 3\}$, and thus there are only four probabilities that can be estimated, $\hat{p} \in \{0/3, 1/3, 2/3, 3/3\}$. All these estimated probabilities are far off the expected probability of $1/2$. This trivial example shows that the real probability $p$ cannot be always known, actually being able to empirically obtain the real probability is an exception rather than a rule. It is because the experiment only is able to estimate a value $\hat{p} = k/n$, which is supposed to be close to the real $p$.

Five simulations of the experiment described above with $n = 1000$ is shown in Table 4.1. It can be seen that even with a large number of experiments ($1000$ experiments), it is not possible to provide an exact estimation of $p$, each one of the five experiments yields different values of $\hat{p}$. Even if we average the probability of the five experiments (in this case $n = 5000$), $\hat{p} = 0.5284$. Thus, providing a fixed value for $p$ without any other information is a partial view of the estimator, and one hardly can do sound claims in base of this estimation [6]. A reference is needed about how far or close $\hat{p}$ is expected to be from $p$. In order to obtain this information we previously have to study the statistical properties of the estimation of a probability, which is a well known problem in Statistics.

## 4.3 Determination of the statistical distribution of SR

Regardless of the particular nature of the EA under study, the estimation of the success probability of an EA consists in running the experiment $n$ times, use a heuristic to identify whether a particular run has been successful, and then count the number of successful runs in generation $i \in \mathbb{N}^+$, $k(i)$. Finally, the estimation is calculated as $\hat{p}(i) = k(i)/n$.

We are usually interested in $p(i)$ when the experiment has finished. So, for clarity and without loss of generality, if the algorithm has been run for $G$ generations, we define SR as $SR = p(G)$. How $p(i)$ depends on time is a different topic that, for the specific case of GP, is addressed in [16].

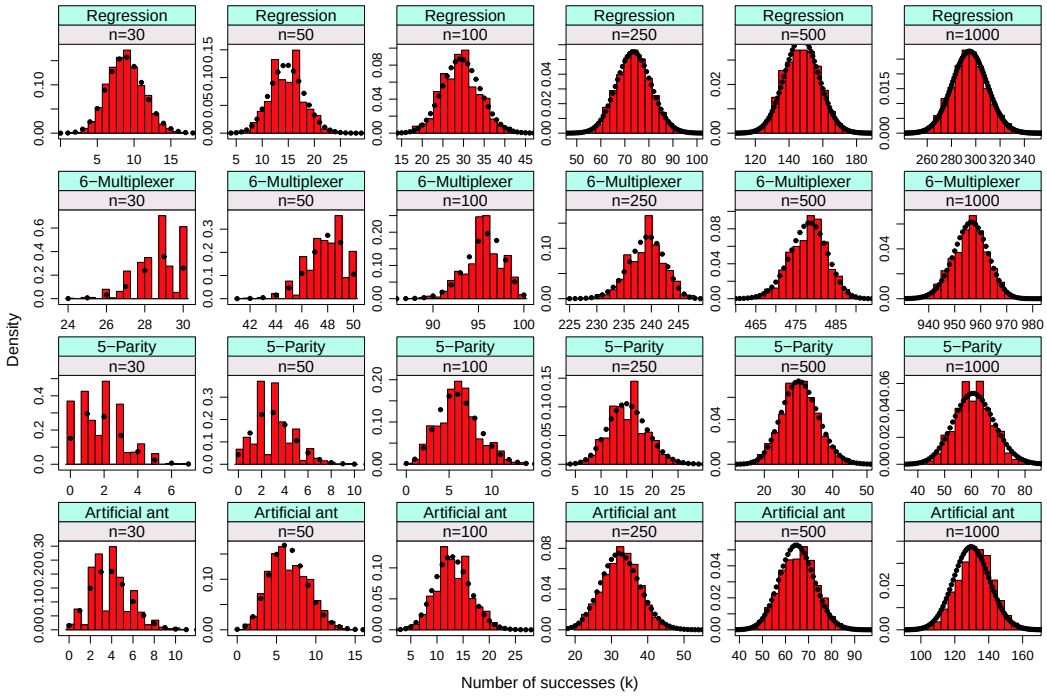


Figure 4.1: Histograms of $\hat{p}_{best}$ for different simulated sample sizes for Santa Fe Trail ($\hat{p}_{best} = 0.13168$), 5-parity ($\hat{p}_{best} = 0.061$), 6-multiplexer ($\hat{p}_{best} = 0.95629$) and regression ($\hat{p}_{best} = 0.29462$). The binomial density distribution Bin($\hat{p}_{best}, n$) is shown overlapped with black points.

If we assume that the experiments are independent, which is not a very restrictive assumption, measuring $\hat{p}$ is equivalent to estimating the number of successes $k$ in $n$ independent experiments. It is well known in statistics that the number $k$ of successes is a random variable described by a binomial distribution, and thus the probability of getting $k$ successes in $n$ trials is given by:

$$Bin(k, n) = \binom{n}{k} p^k (1-p)^{n-k}$$

where $p = k/n$, $C(n,k) = \frac{n!}{k!(n-k)!}$ and $k \in \{0, 1, 2, ..., n\}$.

It is straightforward to deduce the binomial distribution function. Given $n$ experiments, there will be $k$ successes and $n - k$ failures, if the success probability is $p$ then, by definition, the probability of failure is $1 - p$, the probability of getting $k$ successes is $p^k$ and the probability of getting $n - k$ failures is $(1 - p)^{(n-k)}$. Therefore the probability of getting $p^k$ and $n - k$ failures is $p^k(1 - p)^{(n-k)}$. Moreover, the order in which successes appear is not a matter, they can appear in any combination of successes and failures, and there are $C(n, k)$ combinations, so we conclude that the probability of getting $k$ successes in $n$ experiments when the success probability is $p$ is given by $C(n, k)p^k(1 - p)^{(n-k)}$, which is the binomial probability mass function.

So it can be deduced that the probability of getting $k$ successes from $n$ runs in an EA

Table 4.2: Tableau for the problems under study: Artificial Ant with the Santa Fe trail, 6-multiplexer, even 5-parity and symbolic regression without ERC.

| Parameter | Artificial ant | 6-multiplexer | 5-parity | Regression |
|---|---|---|---|---|
| Population | 500 | 500 | 4,000 | 500 |
| Generations | 50 | 50 | 50 | 50 |
| Terminal Set | Left, Right, Move, IfFoodAhead | A0, A1, A2, D0, D1, D2, D3, D4, D5 | D0, D1, D2, D3, D4 | X |
| Function set | Progn2, Progn3, Progn4 | And, Or, Not, If | And, Or, Nand, Nor | Add, Mul, Sub, Div, Sin, Cos, Exp, Log |
| Success predicate | $fitness = 0$ | $fitness = 0$ | $fitness = 0$ | $fitness \leq 0.001$ |
| Initial depth | 5 | 5 | 5 | 5 |
| Max. depth | 17 | 17 | 17 | 17 |
| Selection | Tournament (size=7) | Tournament (size=7) | Tournament (size=7) | Tournament (size=7) |
| Crossover | 0.9 | 0.9 | 0.9 | 0.9 |
| Reproduction | 0.1 | 0.1 | 0.1 | 0.1 |
| Elitism size | 0 | 0 | 0 | 0 |
| Terminals | 0.1 | 0.1 | 0.1 | 0.1 |
| Non terminals | 0.9 | 0.9 | 0.9 | 0.9 |
| Observations | Timesteps=600 | | Even parity | No ERC $y = x^4 + x^3 + x^2 + x$ $x \in [-1, 1]$ |

experiment is described by a binomial distribution. A binomial depends on two parameters, $k$ and $n$, and thus the properties of the estimator of $p$ is independent of the domain and the type of EA used. We can completely characterize the estimator if the number of runs and number of successes are known, which is the common situation in EC. More importantly, the properties of the estimator do not depend on the algorithm internals, following that this is of general application to any EA.

In order to get empirical evidence to support our claim we have selected four GP problems: Artificial ant with the Santa Fe Trail, 6-multiplexer, 5-parity and a symbolic regression problem with no ephemeral random constants (ERCs). These are classical problems proposed by Koza [136] and are widely used by GP literature. We have run the experiments with a standard tree-based GP algorithm using ECJ v18 and its default parameter settings. The main parameters used in the GP executions are shown in Table 4.2.

Experimentation without any trick would require a huge number of runs, so, we used bootstrapping [59]. A large number of $100,000$ runs were executed (this number is reduced to $5,000$ for the 5-parity problem due to computational resource limitations), and its result

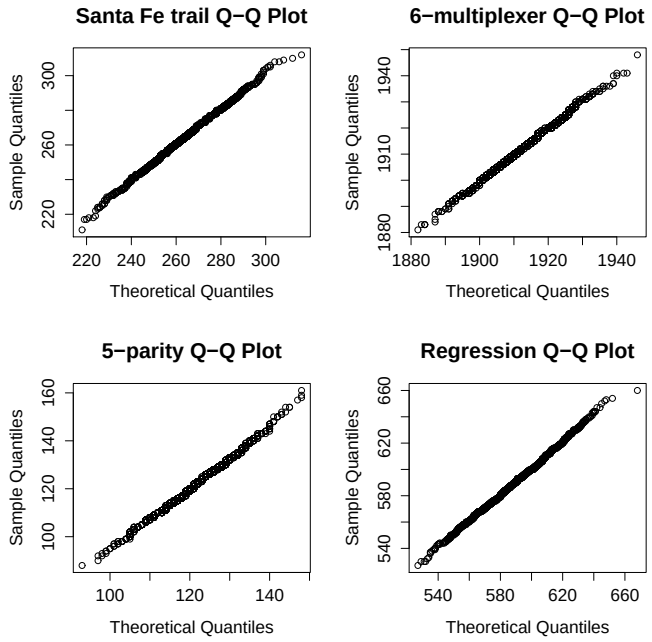


Figure 4.2: Quantile plot of four classical GP problems (Santa Fe Trail, 5-parity, 6-multiplexer and regression with no ERC) against a binomial distribution $Bin(\hat{p}_{best}, 1000)$ ($\hat{p}_{best}$ is, respectively, $0.13$, $0.06$, $0.96$ and $0.29$, see Table 4.3).

stored in a dataset. These datasets are used later to bootstrap $\hat{p}$ with different values of $n$. Since $p$ is not known, we have approximated it with a precise estimation $\hat{p}_{best}$, which used the whole datasets. This precise estimation was used as the real $p$ for comparison purposes. Table 4.3 shows $\hat{p}_{best}$, $k$, $n$ and confidence intervals for $\alpha = 0.05$ and $\alpha = 0.01$ calculated with different methods introduced in the section section.

We first aim to compare graphically experimental results and the binomial distribution $Bin(\hat{p}_{best}, n)$. The procedure is the following one. First, we simulate $2,000$ experiments bootstrapping $2,000$ values of $k$. Each one of these values is calculated resampling with replacement $n$ runs contained in the dataset and counting the number of suscessful runs. This procedure is repeated for each $n \in \{30, 50, 100, 250, 500, 1000\}$. After that, there will be $2,000$ simulated experiments with $n$ runs each one, and a total number of $2,000$ values of $k$. These values of $k$ were represented in an histogram using $n$ as a factor.

The histograms of the four problems under study are depicted in Figure. 4.1. The black points in the figure shows the binomial distribution $Bin(\hat{p}_{best}, n)$. It can be seen that the empirical number of successes follows closely the theoretical binomial distribution in the four problems and all values of $n$, even for the small ones, which is an evidence of binomiality.

In order to provide additional graphic evidence to support our claim, Figure 4.2 shows a quantile plot of the four problems considered in this study. Quantile plots represent the number of successes of $2,000$ bootstrapped values of $k$ with $n = 1000$, as was described above, against the theoretical number of successes obtained from the binomial distribution $Bin(\hat{p}_{best}, 1000)$. The plots show a linear relationship, which suggests the correctness of the

Table 4.3: Best estimation of success probability ($\hat{p}_{best}$), number of successes ($k$) and number of runs ($n$) for the four GP problems under study, and their confidence interval (CI) calculated with the standard (Std), Agresti-Coull (AC) and Wilson (Wil) methods using confidence levels $\alpha = 0.05$ and $\alpha = 0.01$.

| | Artificial ant | 6-multiplexer | 5-parity | Regression |
|---|---|---|---|---|
| $\hat{p}_{best}$ | 0.13168 | 0.95629 | 0.061 | 0.29462 |
| $k$ | 13,168 | 95,629 | 305 | 29,462 |
| $n$ | 100,000 | 100,000 | 5,000 | 100,000 |
| CI $Std_{\alpha=0.05}$ | [0.1295842, 0.1337758] | [0.9550228, 0.9575572] | [0.05436622, 0.06763378] | [0.2917945, 0.2974455] |
| CI $Std_{\alpha=0.01}$ | [0.12892566, 0.1344343] | [0.9546247, 0.9579553] | [0.05228174, 0.06971826] | [0.2909067, 0.2983333] |
| CI $AC_{\alpha=0.05}$ | [0.1295983, 0.1337900] | [0.9550051, 0.9575399] | [0.05468869, 0.06798535] | [0.2918025, 0.2974533] |
| CI $AC_{\alpha=0.01}$ | [0.12894997, 0.1344589] | [0.9545939, 0.9579256] | [0.05283056, 0.07033299] | [0.2909204, 0.2983469] |
| CI $Wil_{\alpha=0.05}$ | [0.1295984, 0.1337899] | [0.9550052, 0.9575397] | [0.05469723, 0.06797681] | [0.2918025, 0.2974533] |
| CI $Wil_{\alpha=0.01}$ | [0.12895008, 0.1344588] | [0.9545942, 0.9579253] | [0.05284989, 0.07031365] | [0.2909204, 0.2983468] |

binomial assumption. Quantile plots for other values of $n$ were depicted (not shown) with the same result.

We performed a fit-of-goodness study with a Pearson's $\chi^2$ test. This test evaluates whether a set of samples comes from a population with a given distribution, a $Bin(\hat{p}_{best}, n)$ in this case. Since the measure of Pearson's $\chi^2$ test is random due to the resampling, all the experiments have been repeated $100$ times and the p-value averaged. The test was performed for $n \in \{15, 30, 50, 100, 250, 500, 1000\}$.

The results of the experiments for the four study cases under study can be found in in Table 4.4. It shows the mean p-value with its standard deviation and their difference. We set the rejection criteria of the null hypothesis (population comes from a $Bin(\hat{p}_{best}, n)$ distribution) as $p - value - sd < \alpha$, i.e., the difference between the p-value and its standard deviation was higher than a certain significance level, let us say $\alpha = 0.05$. Looking at the results shown in Table 4.4 we can observe that almost all the p-values are around $0.23$, but it tends to get lower values when $n$ is higher. Similarly, standard deviations get higher as $n$ increases. Two facts can explain this behaviour. First, the range of values that the random variable $Bin(\hat{p}_{best}, n)$ is wider when $n$ is high, so it is logical that the dispersion of the p-value was proportional to $n$. Secondly, effect size might have a role in the explanation of the results. We should keep in mind that $\hat{p}_{best}$ is just an estimation of the real probability associated to the GP problem, this discrepancy is more apparent when $n$ is high, so it is logical that the p-value got lower values.

Looking at Table 4.4 we only find evidence to reject null hypothesis in four cases, all of them with high values of $n$. The results of the testing in the rest of the cases does not provide enough evidence to lead us to reject our initial hypothesis. Due to the reasons described above, we argue that the few cases where null hypothesis is rejected are type-I errors, concluding that Pearson's $\chi^2$ test supports our claim.

In conclusion, there are strong theoretical reasons to claim that success probability in EAs is a random variable that can be modeled with a binomial distribution. All the experiments carried out in four classical GP problems supports our claim for GP, histograms, quantile plots and Pearson's $\chi^2$ test for fit support the binomiality of the number of successful runs in an EA experiment. Therefore, it seems to be reasonable to assume binomiality until section 4.6, where this issue is resumed and additional evidence provided. One of the most notable consequences of the binomial nature of SR is that the statistical methods developed for binomial can be applied to SR in the context of EA. One of these methods is confidence intervals.

## 4.4 Binomial confidence intervals

Using a binomial distribution to model the SR of EAs entails several benefits, one of them is that all the extense literature about binomials can be applied. In particular, the problem of estimating the SR of an EA can be generalized to the problem of estimating the parameters of a binomial distribution, which has been a subject of intense research in Statistics. Any estimator has a certain associated uncertainty, so, reporting only the value of the estimator provides only a part of the story. It is necessary to provide additional information about that uncertainty. A powerful tool to characterize it is CIs. Our goal is to get a basic understanding

Table 4.4: Pearson's $\chi^2$ goodness-of-fit against a binomial distribution with $\alpha = 0.05$. $1,000$ p-values were calculated, each one with $200$ simulated experiments. Average p-values ($\overline{p-val}$), standard deviation of p-values (sd) and their difference ($diff = \overline{p-val} - sd$) are shown. Data that drives to reject the null hypothesis ($\overline{p-val} - sd < 0.05$) is marked with bold letters.

| | Santa Fe | | | 6-Multiplexer | | | 5-Parity | | | Regression | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N | $\overline{p-val}$ | sd | diff | $\overline{p-val}$ | sd | diff | $\overline{p-val}$ | sd | diff | $\overline{p-val}$ | sd | diff |
| 15 | 0.2275 | 0.0032 | 0.2243 | 0.2206 | 0.0789 | 0.1417 | 0.2211 | 0.0042 | 0.2169 | 0.2331 | 0.0152 | 0.2179 |
| 30 | 0.2303 | 0.0243 | 0.206 | 0.2242 | 0.0060 | 0.2182 | 0.2279 | 0.0041 | 0.2238 | 0.2425 | 0.0203 | 0.2222 |
| 50 | 0.2374 | 0.0197 | 0.2177 | 0.2293 | 0.0053 | 0.224 | 0.2327 | 0.0048 | 0.2279 | 0.2453 | 0.0382 | 0.2071 |
| 100 | 0.2355 | 0.0535 | 0.182 | 0.2342 | 0.0285 | 0.2057 | 0.2383 | 0.0125 | 0.2258 | 0.2316 | 0.0809 | 0.1507 |
| 250 | 0.2397 | 0.1155 | 0.1242 | 0.2420 | 0.0249 | 0.2171 | 0.2300 | 0.0631 | 0.1669 | 0.2132 | 0.1535 | 0.0597 |
| 500 | 0.1885 | 0.1479 | **0.0406** | 0.2326 | 0.0756 | 0.157 | 0.2348 | 0.1044 | 0.1304 | 0.1303 | 0.1629 | **-0.0326** |
| 1000 | 0.1279 | 0.1813 | **-0.0534** | 0.2109 | 0.1301 | 0.0808 | 0.2041 | 0.1006 | 0.1035 | 0.0407 | 0.1006 | **-0.0599** |

of how to use binomial CIs in the context of EAs, with a focus on their properties.

CIs for binomial distribution is a well studied problem due to its wide range of practical applications, so, it is not surprising that there are many methods to calculate binomial CIs [44], and rigorous comparisons have been published [179, 44, 45, 190, 235, 207]. A binomial distribution is fully described by two parameters: the number of trials ($n$), and the number of successes ($k$). Alternately, the success probability $p$ can also be used, which can be directly calculated from $n$ and $k$ simply as $p = k/n$. It is interesting from the perspective of EC because it decouples its study from the particular EA used. Only these two parameters are needed in order to fully describe the statistic properties of the CI, regardless of the internal dynamics of the EA and its particularities. One of the parameters in the binomial distribution, $n$, is usually known by the EA practitioner, while the SR, $p$, is usually unknown and thus it is the parameter that we are usually interested to estimate.

### 4.4.1 Description of the CIs methods under study

There are numerous binomial CIs calculation methods, and including all in this study would be unrealistic, so, we have selected those ones that we consider more representative due to its wide use or its presence in the literature. We have selected four methods: standard, "exact", Agresti-Coull and Wilson. A brief introduction to these methods follows.

**Standard interval**. Also known as *asymptotic method*, *normal approximation* or *Wald interval*. It is the best known, oldest [145] and extended method, even the name represents how extensive the usage is. It is well known that a binomial $Bin(p, n)$, when $np$ is large enough (usually $np > 30$), approximates a normal distribution $N(np, np(1-p))$ (see Figure 4.1). Therefore if the binomial approaches a normal, it is possible to generate intervals with the same method used with the normal distribution [239]. Although this method has been widely reported to suffer several flaws [44, 179, 240, 241], it is widely used due to its simplicity and its presence in basic Statistics books. The standard interval is given by

$$p \pm z_{\alpha/2}\sqrt{\frac{p(1-p)}{n}} \tag{4.1}$$

where $z_{\alpha/2}$ is the upper-$\alpha/2$ critical point from $N(0, 1)$ and whose values can be found tabulated in statistical tables as well as statistical packages. One drawback that the standard interval presents is that this interval cannot be calculated when $p$ is 0 or 1.

**Clopper-Pearson or "exact" interval**. This interval is described as "exact", with quotes, because it is deduced from the binomial distribution. Ironically, despite its name, its discrete nature makes this method unnecessarily conservative, and therefore far from being exact. The limits $[L, U]$ of the "exact" interval [58] are given by the solution to $p$ of the equations $P(bin(n, p_U) \leq X) \geq \frac{\alpha}{2}$ and $P(bin(n, p_L) \geq X) \geq \frac{\alpha}{2}$, which yields the following equations:

$$\sum_{k=0}^{k} \binom{n}{k} p_U^k (1-p_U)^{n-k} = \frac{\alpha}{2}$$

$$\sum_{k=x}^{n} \binom{n}{k} p_L^k (1-p_L)^{n-k} = \frac{\alpha}{2}$$

The solution of these equations is not trivial and can be expressed using the beta distribution as follows.

$$\begin{aligned} L_{CP}(k) &= B(\alpha/2; k, n-k+1) \\ U_{CP}(k) &= B(1-\alpha/2; k+1, n-k) \end{aligned} \tag{4.2}$$

where $B(\alpha; a, b)$ stands for the $\alpha$ quantile of a $Beta(a, b)$ distribution. Sometimes the "exact" interval is expressed as a function of the F-distribution, due to its relationship with the beta distribution:

$$L_{CP}(k) = \left[1 + \frac{n-k+1}{kF_{2k,2(n-k+1),1-\alpha/2}}\right]^{-1}$$
$$U_{CP}(k) = \left[1 + \frac{n-k}{(k+1)F_{2(k+1),2(n-k),\alpha/2}}\right]^{-1}$$

where $F_{a,b,c}$ represents the $1-c$ quantile from the F distribution with degrees of freedom $a$ and $b$.

**Agresti-Coull interval**. Also known as *adjusted Wald*, a term introduced by the original paper of Agresti and Coull [2]. This is a modification of the standard interval where some pseudo-observations are added to (4.1). In this way, instead of calculating the standard interval using $n$ and $p$ computed as $p = k/n$, Agresti-Coull uses $\tilde{p}$ and $\tilde{n}$ calculated as

$$\tilde{p} = \frac{(k + \frac{1}{2}z^2_{\alpha/2})}{(n + z^2_{\alpha/2})}$$

and

$$\tilde{n} = (n + z^2_{\alpha/2})$$

then, the standard interval is calculated as in (4.1), but using $\tilde{p}$ and $\tilde{n}$ instead of $p$ and $n$,

$$\tilde{p} \pm z_{\alpha/2}\sqrt{\frac{\tilde{p}(1-\tilde{p})}{\tilde{n}}} \tag{4.3}$$

It should be pointed out that for a common case where $\alpha = 0.05$, then $z^2_{\alpha/2} \approx 2$ and thus $\tilde{p}$ and $\tilde{n}$ can be simplified to $\tilde{p} = (k+2)/(n+4)$ and $\tilde{n} = (n+4)$. Consequently it is equivalent to adding two failures and two successes. In this way the probability remains unchanged, and the calculus of the CI is the same than the standard intervals, but their properties are significantly improved.

It is interesting to note that the center of the interval is not given by $\hat{p} = \frac{k}{n}$, as usual, but rather by $\hat{p} = (k + \frac{1}{2}z^2_{\alpha/2})(n + z^2_{\alpha/2})^{-1}$, which is not placed in the center of the interval. Nevertheless, as $n$ is increased (and indirectly also $k$), the center of the interval tends to be closer to $\hat{p} = \frac{k}{n}$, so increasing the number of experiments generates more symmetric intervals.

**Wilson interval**. Also known as the *score* method. Wilson interval [247] is derived from a normal approximation as the solutions to the equations $(\tilde{p} - p_0)/\sqrt{p_0(1-\tilde{p})/n} = \pm z^2_{\alpha/2}$ which is given by

$$CI_W = \frac{k + \frac{1}{2}z^2_{\alpha/2}}{n + z^2_{\alpha/2}} \pm \frac{z^2_{\alpha/2}\sqrt{n}}{n + z^2_{\alpha/2}}\sqrt{p(1-p) + \frac{z^2_{\alpha/2}}{4n}} \tag{4.4}$$

The center of the Wilson interval has the same form as Agresti-Coull, so we can point out the same considerations about it. Actually, when $\alpha = 0.05$ Wilson intervals are quite similar to Agresti-Coull.

### 4.4.2 Discussion about CI methods

Many authors have studied the performance of CI methods using rigorous statistical approaches [179, 44, 45, 190, 235, 207]. Brown [44] recommends, for small $n$ (40 or less), Wilson or Jeffreys (a variation of Bayes intervals, not covered here) methods, while for large $n$ values (more than 40) he recommends also Agresti-Coull. Similarly, Piegorsch [189] remarks that while Wilson and Jeffreys perform better when $n < 40$, the rest of methods are similar for higher values of $n$.

Some GP studies have been focused in the more specific problem of estimating the computational effort, [240, 241, 180], all of them have noticed the poor performance of normal approximation, and recommend the use of Wilson. These studies apply several CIs methods to computational effort, nevertheless they use a pure experimental approach, without a theoretical or statistical justification to support the methods used. It is not considered that some of the CIs studied, such as Wilson, are supposed to be used with binomial distributions.

We aim to study the performance of the most significant binomial CI methods from a systematic, general and problem independent point of view, and how its performance depends on $p$ and $n$. Once the behaviour of the CIs was understood in terms of $p$ and $n$, it is easy to extrapolate the results to a EC experimental context.

## 4.5 Study on some confidence interval methods performance

This section, inspired by [44], analyzes the performance of some CI methods. We are interested in showing the relationship between the two parameters of a binomial distribution and how they influence the performance of the CI. We use two related metrics to measure the performance of the CI methods, the coverage probability and the interval width.

On the one hand *coverage probability* (or CP) is defined as the probability of a CI to contain $p$, more formally, $CP = P(L \leq p \leq U)$. It is worth noting that increasing the CP of an interval is trivial, just increasing its width. Furthermore, the coverage of the CI $[0, 1]$ is always $1$ because $p$ must be contained in that interval by definition. On the other hand CI width (or *CIW*) is defined as the difference between U and L, $CIW = U - L$. Of course, a tight interval is better than a wide interval, given that both have the same CP.

CP and CIW are closely related. There is a trade-off between CP and CIW, so CI methods have to find a balance between them. A good CI is not that one with a CP next to $1$, but rather a tight interval with a CP close to the nominal coverage $1-\alpha$, i.e., $P(L \leq p \leq U) \approx 1-\alpha$. If an interval achieves a CP higher than the nominal one is at a cost of a wider interval. In terms of EA experimentation, such a conservative method would lead, for example, to a higher difficulty to detect significant differences between the SR of two algorithms. Understanding the properties of CP and CIW might lead to designing better experiments related to SR and composed measures such as computational effort.

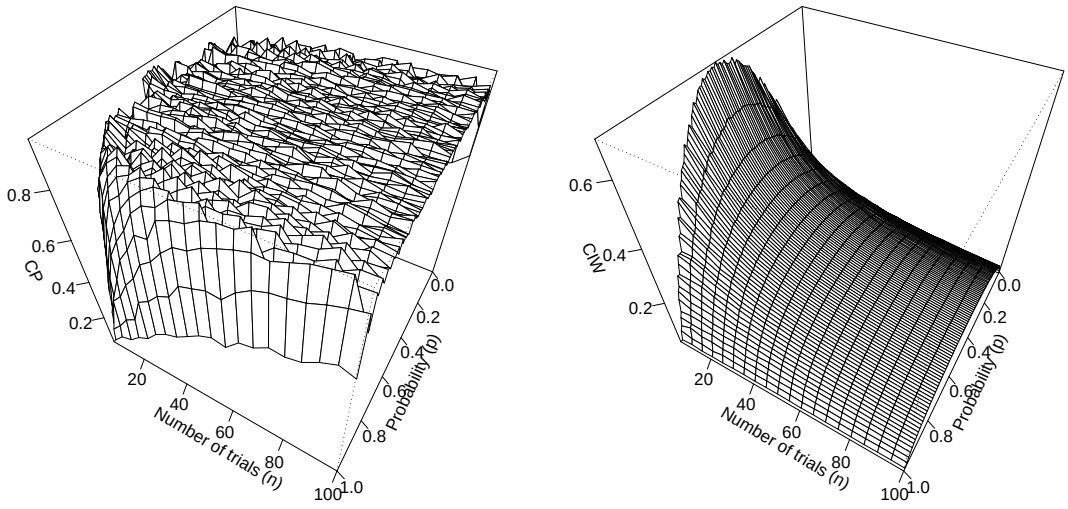


Figure 4.3: CP (left) and CIW (right) for various sample sizes ($n$) and SR ($p$) for standard CI with confidence level $\alpha = 0.05$. Similar shape and fluctuations are found in the rest of the methods.

### 4.5.1 CIs performance overview

The performance of an interval depends on the binomial parameters, but these parameters are not independent. In order to get an initial glimpse to this question we need a huge set of EA experiments as well as well as a strict control on the SR, which is quite difficult to achieve in real EAs. So, instead of running EAs, we have simulated them.

Since parameters $n$ and $p$ fully describe the binomial CI of an EA, we do not have to run the algorithm. From the SR point of view, the result of an EA experiment is binary. Therefore we have characterized pseudo-EA runs with a set of labels, "success" or "failure". For each probability $p \in \{i/2000 \forall i = 1, ..., 2000\}$, we generated a dataset with $100.000$ labels "success" with probability $p$ and "failure" with probability $1-p$. Once the dataset was created, we took $n$ pseudo-experiments $2,000$ times, with $n = 5, 6, ..., 100$, and calculated $\hat{p}_i$ for each set of experiments. In summary, we bootstrapped $\hat{p}$ using $2,000$ resamples for several combinations of $p$ and $n$. The CI with $\alpha = 0.05$ was calculated using each method under study and each combination of $p$ and $n$. Finally, CP and CIW were calculated. In this way we simulated the execution of EAs varying number of runs and probabilities, yielding a total of $95 \times 2,000 \times 2,000 = 380,000,000$ simulated EA runs.

The relationship between CIW, $p$ and $n$ can be seen in Figure 4.3 (right), where the CIW of standard CIs with $\alpha = 0.05$ is depicted. The shape of the figure is the same for the rest of the methods under study, so we only show the diagram of one method. CIW is symmetric for the plane $p = 0.5$ due to the fact that these methods are equivariant, their limits for $(n-k)/n$ are complements of those for $k/n$ [179]. The plane $p = 0.5$ defines the symmetry of the figure as well as the maximum of CIW. CIs are wider when SR is close to $0.5$, alternatively the closer is $p$ to its boundaries $0$ and $1$, the tighter is the interval. It is explained by the constraints that the boundaries introduces to the calculation of the interval. Looking at the

behaviour of CIW with $n$, we can observe that CIW is monotonically decreasing, whether the number of runs is increased there is additional information that is used to build tighter intervals. Of course, the price of such improvement is an increase of the number of runs and computational resources.

Figure 4.3 (left) depicts the coverage of standard CIs for several values of $p$ and $n$. It can be seen that Figure 4.3 (left) depicts a rather chaotic behaviour, with many peaks and valleys without a clear pattern. We will show later that this behaviour is not actually chaotic but rather the low resolution of Figure 4.4 which hides some phenomena very characteristic of binomial CIs. It will be analyzed in detail in the next section.

Some patterns can be found in Figure 4.3 (left). In particular, it is pretty clear that low coverage is associated to a low number of trials or a SR close to $0$ and $1$. This fact is also observed in the other methods and is intrinsic to the nature of the binomial distribution. However, there are quantitative differences among the methods. In the case of the standard interval the effects of low $n$ values are dramatic because the normal approximation is no longer valid. This behaviour is consistent with the one found in CIW: the wider is the CI, the less restrictive it is, and thus it is more likely that the real $p$ was contained in the interval.

The rest of the methods present the same high level behaviour described above, so all of them share some common properties which seem to be intrinsic to the problem, however, their performance differs significantly when analyzed in detail. It worths exploring this issue.

### 4.5.2 Coverage of CIs

An analysis of CP shows some remarkable facts. Figure 4.4 represents the coverage surface of the methods under study for $n$ between $20$ and $200$ in steps of $1$ and $p$ takes $2,000$ values between $0$ and $1$. CP was calculated using R's function `binom::binom.coverage()`.

Figure 4.4 shows that the chaotic behaviour of CP seen in Figure 4.3 (left) actually follows a pattern with symmetry in the axis $p = 0.5$. The low resolution and sampling noise of Figure 4.3 (left) hid this pattern. As was previously seen, regardless of the used method, there are some areas with poor performance in terms of CP placed on the boundaries of $p$ and low values of $n$. Coverage is particularly low in the bottom corners of the graph, where both, $n$ and $p$ have negative influence on CP. This is a low coverage area, where simply there is not enough information to define reliable intervals. However, the coverage of the standard interval is dramatically poor CP in these corners in relation to the rest of the methods. Coverage of standard intervals achieves extremely low values when $n < 15$ and $p < 0.1$.

A new, and striking, phenomenon that was not observed previously in Figure 4.3 (left) is the presence of oscillations in the coverage regardless the CI method used. These oscillations are a well known phenomena [44, 2] and they are generated by the discreteness of the binomial distribution. The magnitude of the oscillations has a great impact in the overall performance of the CI method. Oscillations appear in Figure 4.4 as waves whose magnitude is inversely proportional to the value of $n$. The magnitude of the oscillations also depends on $p$, and it is, like CP and CIW, symmetrical with respect $p = 0.5$.

Despite the fact that CP presents oscillations regardless the method being used, their magnitude changes. It is clear that, for instance, standard intervals coverage oscillates strongly when $n$ is low in comparison to the rest of the methods. Wilson and "exact" methods contains coverage oscillations less pronounced than those in the standard method.

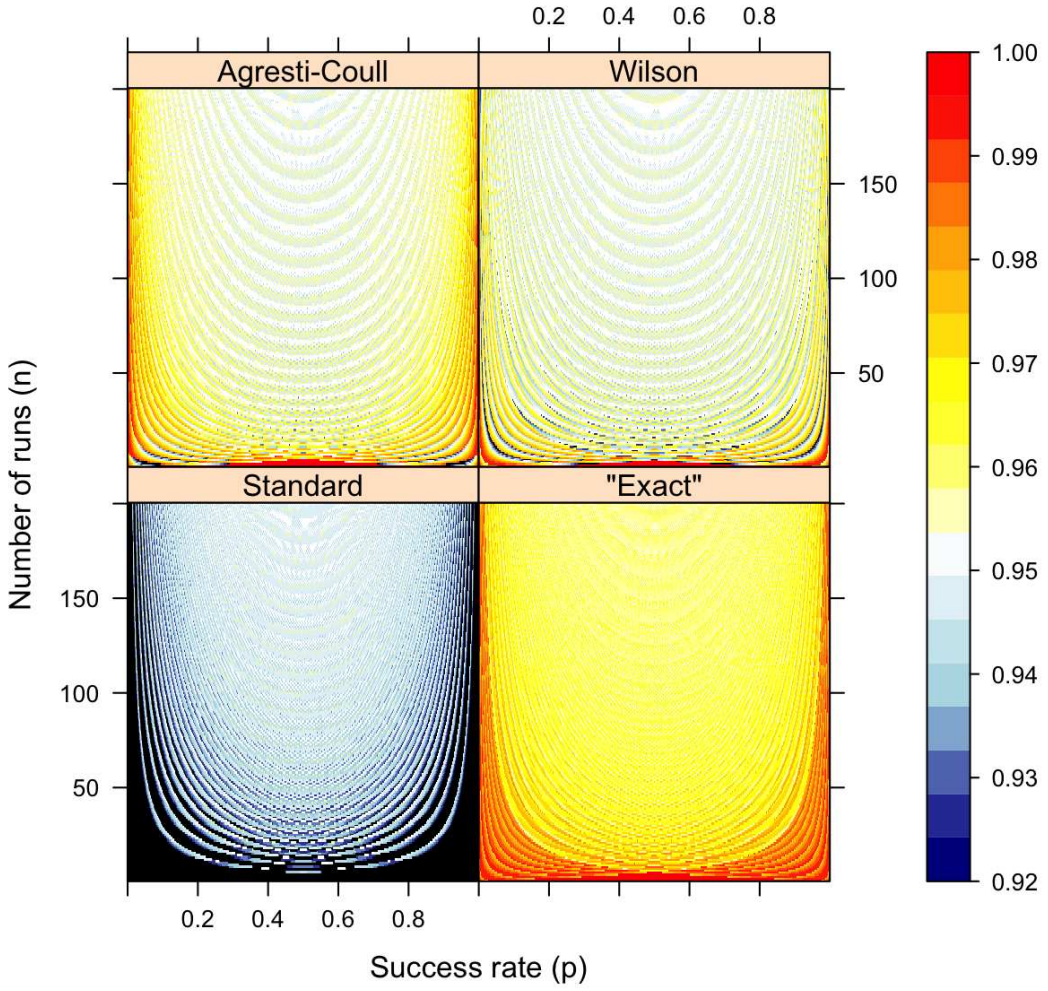


Figure 4.4: Dependence between coverage, $n$ and $p$ for CI methods under study: standard, "exact", Agresti-Coull and Wilson, all calculated with $\alpha = 0.05$. X-axis represents success probability, $p$, while y-axis represents the number of runs, $n$. Coverage values lower than $0.92$ have been represented in black, while coverage that equals the nominal value $0.95$ is ploted in white.

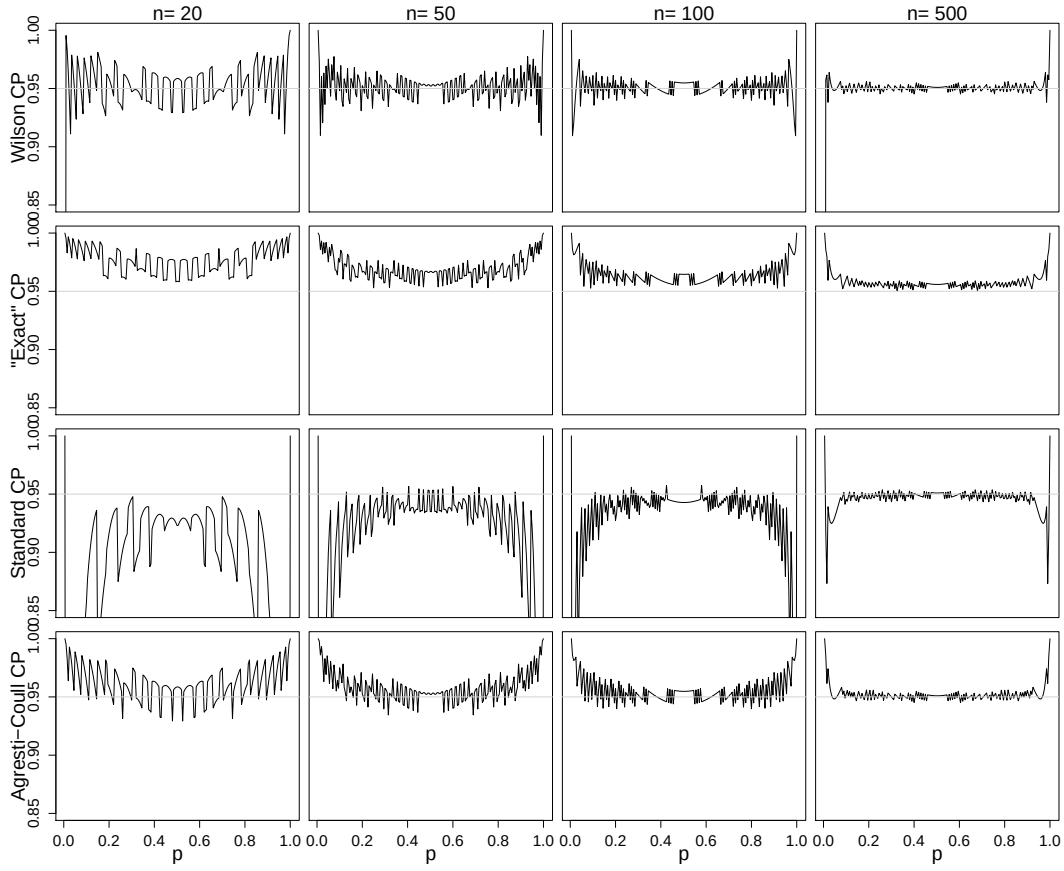


Figure 4.5: Comparison of CP for different CI methods (Wilson, “exact”, standardand Agresti-Coull) and number of samples ($20$, $50$, $100$ and $500$). The nominal coverage, $\alpha = 0.95$, is represented with an horizontal grey line.

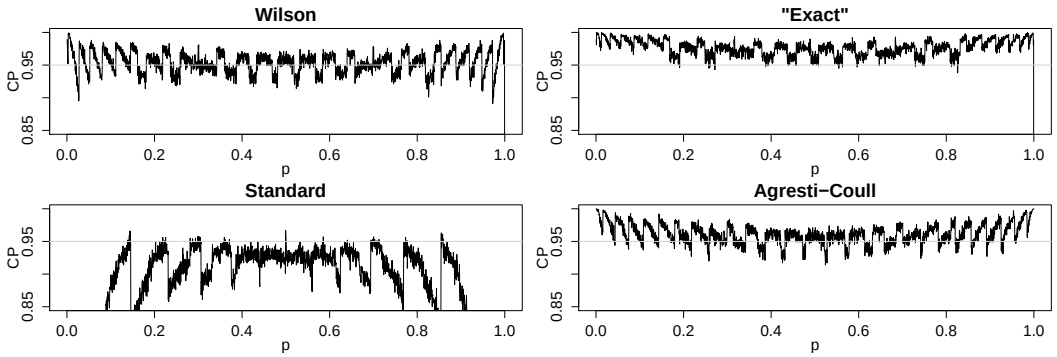


Figure 4.6: Empirical CP measured for simulated GP experiments when $n = 20$ and $\alpha = 0.05$. This coverage has the same shape than the nominal coverage shown in Figure 4.5, but contamined by noise generated by the measurement.

It is important to verify how close is CP to the nominal coverage. This point can be appreciated more clearly in Figure 4.5, which shows several "cuts" of the coverage surfaces at some values of $n$. In this way we obtain a detailed view of the oscillations. An ideal CI method would generate intervals with a CP equal to the nominal one, nevertheless it is clear looking at Figure 4.5 that the real coverage is far from being equal to the nominal one, $0.95$ in this case.

There are some common characteristics of the oscillations for all the methods under study. The magnitude of the oscillations and its shape is directly related to $n$. When $n$ is large, for instance $500$, CP gets a rather flat shape with small oscillations and CP gets pretty close to the nominal coverage. Looking at Figure 4.5 we conclude that, in the same line than the ones reported by [44], when $n$ is large enough the differences of coverage properties of the CI methods under study are not significant.

Each method exhibits some particularities in the behaviour of their coverage. "Exact" intervals coverage is always higher than the nominal one. This conservatism produces wider intervals, as will be shown later. On the contrary, standard intervals coverage is lower than the nominal coverage, indeed when $n$ is low, the coverage is much lower. Even when $n = 100$, which is a relatively high number of trials, its performance in the boundaries of $p$ is poor compared to the other methods. Agresti-Coull interval does not exhibit such a clear behaviour. It has areas where CP is higher than the nominal one, and other areas where CP is lower, however it tends to be more conservative in the boundaries of $p$. Finally, Wilson intervals show good coverage properties close to $1 - \alpha$ and it is neither conservative nor liberal.

It is worth comparing the exact CP with those obtained in the simulated executions of GP. Figure 4.6 depicts coverage diagrams obtained by the experiment described in the beginning of the section for $n = 20$. It can be seen that Figure 4.6 fits nicely with the analytical coverage represented in Figure 4.5, with the only exception of a high frequency noise produced by the sampling. Given that the CP diagram shown in Figure 4.6 is itself the estimation of a probability, the existence of this noise is now surprising. In any case, it seems clear that the shapes of both diagrams follow the same pattern, and thus the underlying probability distribution is likely to be the same, i.e., a binomial distribution, providing additional support to

our hypothesis.

### 4.5.3 Average CP

Some characteristics of the coverage properties can be better viewed using average values of CP, as they are shown in Figure 4.7. It shows the CP averaged for $1,000$ values of $p$ between $0$ and $1$ (top) and values of $n$ between $5$ and $49$ (bottom). We used the `binom::binom.coverage()` function from the R `binom` package.

Figure 4.7 (top) shows how for low number of runs average coverage is degradated in all the methods, nevertheless, it does not affect equally to all. Standard method has very poor average performance when $n$ is low. On the contrary, "exact" method presents a rather high average CP for small number of runs, which is consistent with the conservative behaviour of this method previously observed. Agresti-Coull method is also quite conservative, however less than the "exact" method, its average CP is close to the nominal one. Finally, Wilson CIs have an outstanding performance with a low number of runs. When $n$ is very low, arround $5$, its average CP is close to the one in the "exact" method, nonetheless it dramatically decreases for higher number of runs, achieving an average CP very close to the nominal one.

It is interesting to observe the average CP when the number of runs is high. Figure 4.7 (top) shows that increasing the number of runs tends to reduce the difference among the methods, but not to the point of diluting all the differences. Even for a relative high number of runs ($n \approx 100$), the standard method has a disappointing performance with an average CP much lower than the nominal one. A glance to Figure 4.5 shows that the low average CP is due to its poor performance in the boundaries of $p$. In opposition to the standard method, the "exact" method tends to generate conservative intervals even with high number of runs. Agresti-Coull and Wilson are the methods that are closest to the nominal coverage when $n$ is high, with a small advantage to Wilson.

Figure 4.7 (bottom) adds a complementary perspective where CP has been averaged for values of $n$ between $5$ and $49$. Standard method has very poor coverage properties, dramatically poor when $p$ is close to $0$ or $1$. To be honest, it should be pointed out that $n$ values used to average is rather low, just where its performance is worse. The conservatism of the "exact" method is evident observing the figure, this method generates the highest CP. This property makes it more difficult to find differences among algorithms, however it minimizes finding false differences, which might be of interest depending on the experimentation goals. Close to $p = 0.5$ all the methods have similar average CP, with the only exception of the "exact" method, again, with a high CP in comparison with the rest of the methods. The method whose average CP is closest to the nominal coverage is clearly Wilson's method, achieving a quite flat average coverage plot, even in extreme values of $p$, where the average CP is slightly increased.

### 4.5.4 Average CIW

The overall picture of how CI methods perform should be completed looking at CIW. Unlike CP, CIW has not a random nature, given a certain $n$ and $p$, the exact value of CIW can be determined. Average CIW values were calculated using $n$ (Figure 4.8 top) and $p$ (Figure 4.8 bottom) as independent variable. Many properties of the CI methods are equivalent or com-

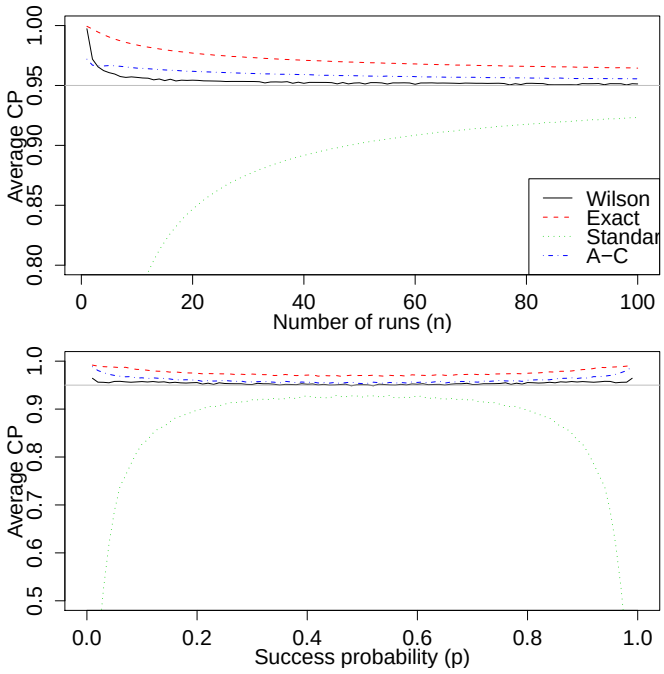


Figure 4.7: Comparison of average CP for different CI methods with fixed success probability $p$ (top) and fixed number of runs $n$ (bottom) (source: [44]).

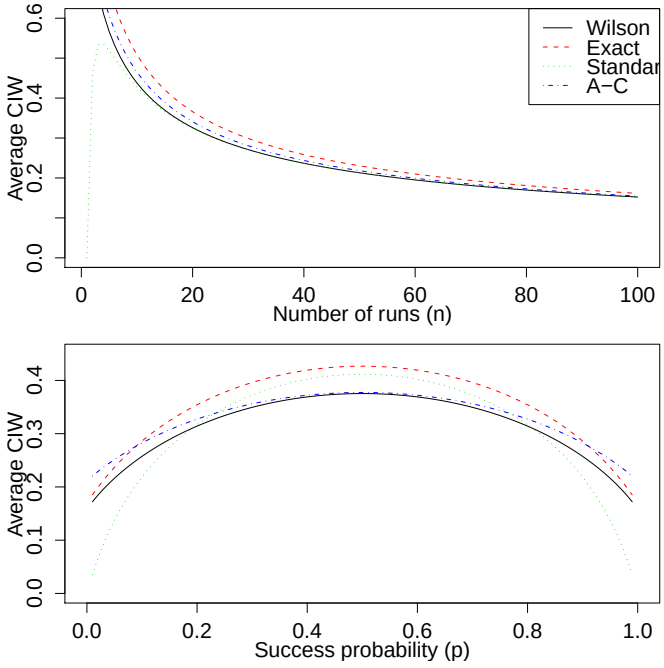


Figure 4.8: Comparison of average CIW for different CI methods with fixed success probability $p$ (top) and fixed number of runs $n$ (bottom) (source: [44]).

plementary to those observed for the average CP because of the close relationship between CP and CIW. The most notable differences among the CI methods are found when $n$ is small and $p$ is close to its boundaries, just like CP. Similarly, when $n$ is large, average CIW tends to be rather similar in all the CI methods. The same happens when $p$ is close to $0.5$ between Wilson and Agresti-Coull.

Figure 4.8 (top) shows that there is a clear relationship between the average CIW and the number of trials: the smaller $n$ is, the wider the interval is. Indeed it does not follow a linear relationship: when $n$ is small adding few runs dramatically reduces CIW, but the effect of increasing $n$ is less notable when $n$ is greater, until a point where increasing $n$ does not pay off due to the limited improvements in CIW.

CIW graphs explain some facts about coverage properties. The high CP found for the "exact" method has its counterpart in CIW; high average CP is achieved at a cost of wider average intervals. This fact can be observed for almost all the values of $p$ and $n$ shown in Figure 4.8 (top and bottom). The normal approximation yields slightly wider CIs, except in case of low $p$ values, just where CP is much worse. Wilson shows an excellent performance from the average CIW point of view with tight intervals.

### 4.5.5 Discussion of the results

Looking at the results shown in this section, we suggest not using standard method in any case, its performance in terms of CP and CIW ranges from mediocre (when $np \gg 5$), to very poor ($np \leq 5$). The simplicity, availability and presence in the literature is a point to take into account in favor of Wilson's method. In any case, there is not a method with better CP and CIW in absolute terms. In average terms, Wilson seems to be a good election, but in order to be strict selecting the method with the best performance for an EA, we suggest to analyze first the area of the binomial parameter space in which SR would likely to be placed, and then look at CP and CIW of the methods in that area to select the method with better performance.

Another important subject to take into account when selecting a CI method, is the particularities of the experimentation. It might be important, for instance, being able to detect differences of SR between two EAs with a high level of confidence, avoiding type-II errors as much as possible. In this case using the "exact" method might be interesting, at the price of making it harder to find these differences.

In any case, when the SR is very low, and it is not possible to run a large number of runs, the methods described in this chapter are no longer recommended. When this situation is found, it is better to approximate the binomial $Bin(n, p)$ with a Poisson distribution with expectation $\lambda = np$ [109, 73].

We have provided so far some theoretical and empirical evidences that support the binomiality of SR, as well as a glance to the performance of CIs. A natural question arises at this point: Does the behaviour of CI performance studied above also describe CI performance in real EA experiments?. It is clear that in case SR was binomial its CIs would have the same performance, suggesting an affirmative answer, however more direct evidence is actually desirable.

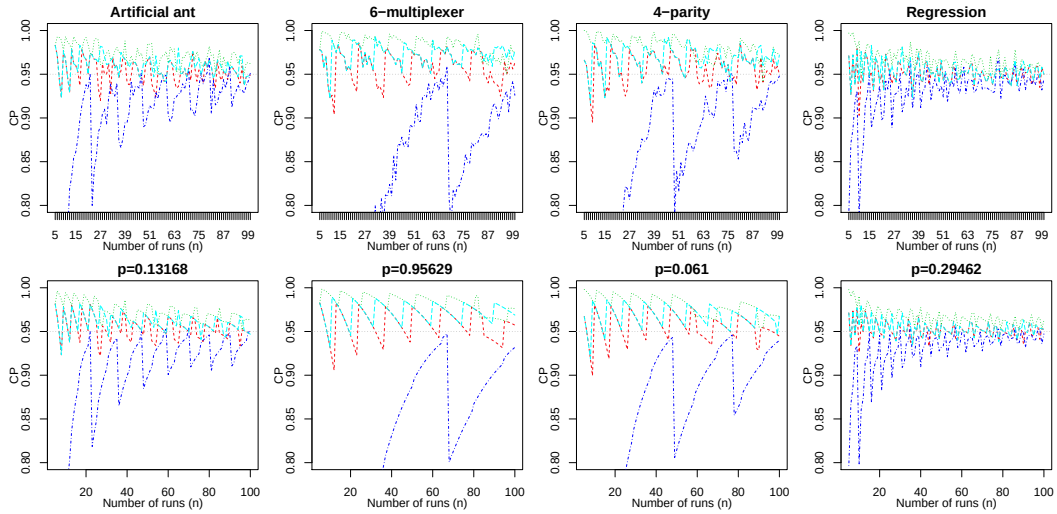


Figure 4.9: Empirical CP for the four domains (top) compared to CP of the binomial distribution (bottom) with the same SR ($\alpha = 0.05$).

## 4.6 Empirical study on the binomial CIs in tree-based GP

We aim to test if the theoretical CP and CIW curves shown above are related with those ones obtained from real GP problems. So, CP and CIW curves have been generated using the experimental setup described in section 4.3 with its four GP problems: Artificial ant with the Santa Fe trail, 6-multiplexer, even 6-parity and symbolic regression. CP and CIW were calculated using the same experimental procedure described in section 4.5.2, however, instead of using a dataset composed by pseudoexperiments, real GP experiments was used to generate $2,000$ intervals and calculate CP.

Figures 4.9 and 4.10 compare, respectively, CP and CIW of the four problems under study (first row) with the theoretical CP and CIW of a binomial $Bin(n, \hat{p}_{best})$ (second row). This diagram is visually very similar to the theoretical ones shown in Figure 4.8 and 4.7. This result supports our hypothesis than $p$ fits actually a binomial distribution.

## 4.7 Sample size determination of confidence intervals

We are interested in getting precise measure of SR. Such an objective is rather simple to achieve, just increasing the number of trials, however, the computational costs of running an EA might be high, so increasing $n$ without a well founded criteria may not be a practical solution. It would be desirable using a well grounded mechanism to set *a priori* the sample size needed to get intervals of the quality desired by the practitioner or researcher. Such a mechanism would, on the one hand, avoid wasting unnecessary computational resources running the EA just the number of times to reach a certain quality, and, on the other hand, provide a solid methodology to set the number of runs.

When someone calculates the CI, they know the number of experiments that have been run, and the number of successes that have been achieved. However, we can state the problem

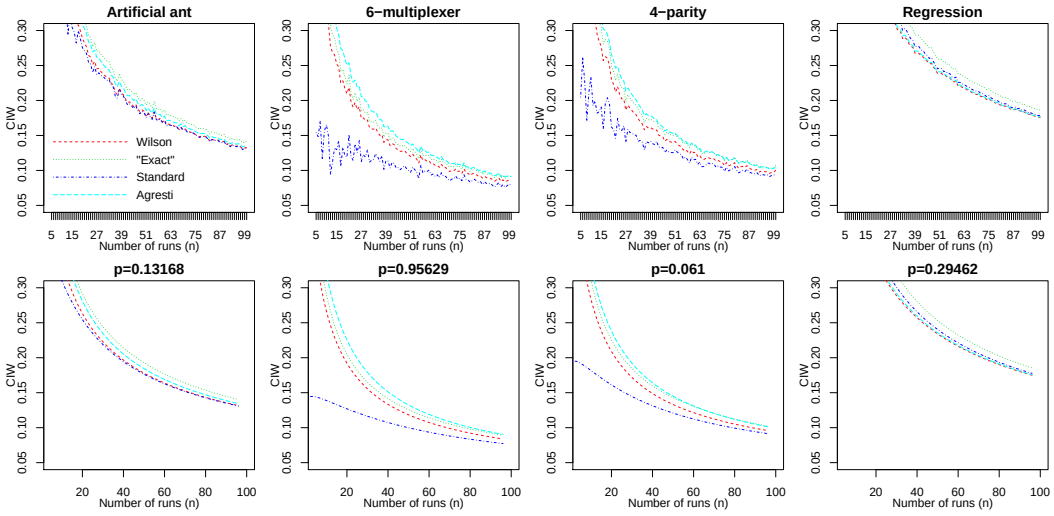


Figure 4.10: Experimental CIW for the four domains (top) compared to CIW of the binomial distribution (bottom) with the same SR ($\alpha = 0.05$).

from another point of view. Instead of estimating $p$ (or $k$, which is the same problem), we could make an initial rough estimation of $p$ with a small number of runs, let's call it $p_0$ instead of $\hat{p}$, set a certain CIW for the CI, and then obtain $n$ from the equations of the CIs. This approach was proposed by Piegorsch [189] to estimate the number of samples needed to obtain intervals with a certain CIW. In this section we summarize some of his results. There are other approaches, especially from the bayesian perspective, and a number of studies have been published [198, 214, 171] addressing this topic from a statistical point of view.

Piegorsch describes in [189] a method to calculate the sample size for the standard, Agresti-Coull, Wilson and Jeffreys methods. For simplicity, instead of using the CIW, he used the half of the interval, $\varepsilon = CIW/2$. For the Standard method, we can state that the half of the interval is, using eq. (4.1), $\varepsilon = z_{\alpha/2}\sqrt{p_0(1-p_0)/n}$, solving that equation for $n$ is straightforward, yielding

$$n_s = \frac{z^2_{\alpha/2} p_0(1-p_0)}{\epsilon^2} \tag{4.5}$$

The same procedure can be used for Agresti-Coull, the half of the interval in this case is given by eq. (4.3) as $z_{\alpha/2}\sqrt{\tilde{p}(1-\tilde{p})/\tilde{n}}$ and solving for $n$ we obtain eq. (4.6),

$$n_{AC} = \frac{z^2_{\alpha/2} p_0(1-p_0)}{\epsilon^2} - z^2_{\alpha/2} = n_s - z^2_{\alpha/2} \tag{4.6}$$

It should be mentioned that Piegorsch does not recommend using (4.6) with less than $40$ samples. Similarly to the standard and Agresti-Coull intervals, Wilson sample size is the solution of $n$ when the half of the interval of (4.4) equals $\varepsilon$, yielding the following expression:

$$n_W = z^2_{\alpha/2} \frac{p_0(1-p_0) - 2\varepsilon^2 + \sqrt{p^2(1-p_0)^2 + 4\varepsilon^2(p_0 - \frac{1}{2})^2}}{2\varepsilon^2} \tag{4.7}$$

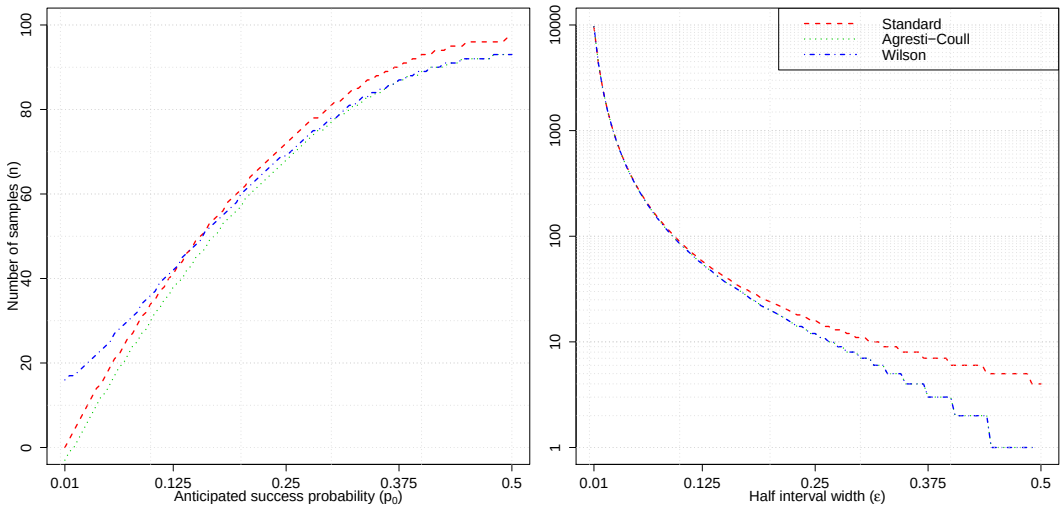


Figure 4.11: Sample size for standard, Agresti-Coull and Wilson CI methods for several anticipated success probabilities $p_0$ with $\varepsilon = CIW/2 = 0.1$ (left) and different half interval widths when $p_0 = 1/2$. Notice the logarithmic scale in the latter one. Unless for low values of $p_0$, the sample size needed by Wilson is lower than for the other methods. Confidence level is set to $\alpha = 0.05$.

In-depth discussion of the equations described above is out of the scope of this dissertation, but it can be found in [189]. However, it is worth a brief discussion to bring some interesting points to the EC community. Comparing (4.5) and (4.6) it is clear that since $z^2_{\alpha/2}$ is positive, Agresti-Coull always requires less samples than standard method to achieve an interval of the same length. It is interesting to point out that when $p_o = 1/2$, $n_W$ equals $n_{AC}$, a fact consistent with the relationship between Agresti-Coull and Wilson interval previously shown.

A major concern to calculate the sample size method described in this section is the measure of $p_0$, which is, itself, the problem we face when calculating binomial CIs. A conservative solution to deal with this problem without estimating $p_0$ is to use the fact that CIs are widest when $p = 1/2$. If we set $p_0 = 1/2$ (the worst case) it is guaranteed that the resulting sample size will generate intervals, at least, of the desired half-length $\varepsilon$. It could be better understood looking at Figure 4.11 (left), this figure represents the sample size as a function of $p_0$ when $\varepsilon = 0.1$, i.e., an interval of the form $[\hat{p} - 0.1, \hat{p} + 0.1]$. The same behaviour is observed for different values of $\varepsilon$. Figure 4.11 (left) clearly shows that, no matter which method is used, the value of $p_0$ that originates the highest sample size is $1/2$, so it is a good conservative election when there is no information about its value.

Observing Figure 4.11 (left) with more detail is interesting. Agresti-Coull requires always less samples than the standard CI, and this method takes the same sample size than Wilson next to $p_0 = 1/2$. Figure 4.11 (left) could lead to mistakenly conclude that Agresti-Coull is the best choice to reduce the sample size if $p_0 < 0.3$ and $\varepsilon = 0.1$. To get a complete picture, CP should also be considered. From the CIW point of view it is clear that Agresti-Coull would be the best choice, but looking at CP we can see that the small sample size is

at a cost of a bad CP performance. So in this case Agresti-Coull needs fewer runs, but it generates less reliable intervals. In the low coverage region (small $p_0$) Wilson has a slightly better CP performance, at a cost of a higher sample size, as can be seen in Figure 4.11 (left).

When $p_0$ is unknown a conservative value $p_0 = 1/2$ might be a good choice. Figure 4.11 (right) represents the sample size as function of $\varepsilon$ in this situation. The sample size dramatically increases with the inverse of $\varepsilon$. This behaviour is explained by the presence of $\varepsilon$ in the denominator of (4.5), (4.6) and (4.7), and is logical, if we desire a tighter CI, we would need more information to build it, which is translated into more runs. As it was previously noticed, Agresti-Coull and Wilson intervals generate exactly the same sample size because when $p = 1/2$ both intervals are actually the same. Finally, it is interesting to notice that for wide intervals the sample size is as low as 1. When $\varepsilon \lesssim 0.5$ the interval takes the form $[L, U] \approx [0, 1]$, and therefore it almost covers all the possible values of $p$. In other words, the interval is so wide that it does not need much information to be constructed, yielding extremely low sample sizes.

In conclusion, the selection of the sample size has to take into account several criteria, some of them mutually exclusive, so a compromise is needed. Usually the goal is to obtain an interval with a certain CIW (or $\varepsilon$) with the lowest number of runs to save computational resources. Eqs. (4.5), (4.6) and (4.7) provide a mean to calculate the number of runs required to achieve a CI with a given CIW regardless of the associated CP. Figure 4.4 provides a mean to estimate the expected CP for the calculated sample size for $\alpha = 0.05$. In case that CP is not the expected one, it is necessary to increase the number of samples until CP achieves an acceptable value. So, $n$ is determined by the maximum sample size between the desirable CP and CIW.

## 4.8 Conclusions

In this chapter we have provided theoretical and empirical evidence suggesting that SR in an EA can be modeled with a binomial distribution. Hence, the extensive literature about binomials can be applied, including CIs, determination of the sample size, hypothesis test for difference between proportions and so on. An important problem related to EC experimentation is the measurement of SR. Due to the binomial nature of SR, its estimation is the same problem that the estimation of the parameters of a binomial distribution, and their statistical properties do not depend directly on the internals of the algorithm.

CIs are a statistical tool with a potential role when studying the performance of an EA. We have described some binomial CI methods with some of their main properties, drawing a picture useful to generate more robust experimental designs in EC. It was found that Wilson is the method that provides better average performance, even for low number of samples and SR next to the boundaries, nonetheless there is no method with the best CP for all the parameter space. Depending on the nature of the experimentation, other methods might be interesting due to their properties, such as Agresti-Coull or the ”exact” method when a conservative method is needed. In any case, experiments shown in this chapter and related literature strongly discourage the use of the standard interval. Despite the method chosen, we encourage EA reporting $n$ and $k$, as well as the interval, to ease further statistical manipulation and comparability of the results.

Finally, we reported some guidelines to select the number of runs to generate SR intervals with a certain expected performance in terms of CIW and CP. Of course, SR is not usually the only measure that is taken from an EA, it only gives a partial view of the algorithm performance, that, with other measures, helps to understand the behaviour of an EA.

Along this chapter, we have assumed than an EA is something static, i.e., we have intentionally excluded time in this study, considering the result of the algorithm when it has finished its execution. It takes sense given the different statistical properties of SR and success probability, but that only is a part of the story. Understanding how the success probability behaves along the time is something basic to answer the research questions that drive this dissertation. In particular, we have to understand the run-time behaviour of the success probability in order to develop an analytical model and use it to characterize the Koza's computational effort. That is the goal of the next chapter.

# Chapter 5

# Run-time analysis of tree-based Genetic Programming applied to model the success probability

*Organic life beneath the shoreless waves*
*Was born and nurs'd in Ocean's pearly caves;*
*First forms minute, unseen by spheric glass,*
*Move on the mud, or pierce the watery mass;*
*These, as successive generations bloom,*
*New powers acquire, and larger limbs assume;*
*Whence countless groups of vegetation spring,*
*And breathing realms of fin, and feet, and wing.*
*The Temple of Nature. Erasmus Darwin*

We need a analytical model of the success probability in order to characterize the error associated to the measurement of the Koza's computational effort. This chapter is devoted to develop such analytical model. The model we propose is based on a decomposition of the success probability into two terms. The first term is static and models the success probability at the end of the execution of the algorithm. This is a binomial random variable and its statistical properties were previously studied in the chapter 4. The second term models the variation of the success probability with time, and thus, it depends on the run-time behaviour of the algorithm, which is unknown. In order to determine how is that run-time behaviour, we perform an experimental analysis of the run-time required to find a solution, that we name *run-time to success*.

As a consequence of the run-time analysis performed to several classical GP problems, we find that the run-time to success follows a complex pattern in function of the problem difficulty and the parameter setting. In general, the run-time to success seems to fit well a lognormal distribution, however, in difficult boolean problems this claim does not hold, and

in that case the right tail of the run-time to success is exponentially distribution. To complete the picture, when the selection is performed at random, the run-time behaviour is described by a Weibull distribution. These results are used to include the lognormal distribution in the model of success probability which is used in chapter 6 to characterize the estimation error associated to the Koza's computational effort.

Even though the experimentation carried out in this chapter only involves tree-based GP, there are reasons to suspect that the results are, at least partially, generalizable. One of these reasons can be found in the literature dedicated to the run-time analysis of several metaheuristics. In an attempt to generalize the results, we propose a theoretical model based on Markov Chains, and demostrate that in ausence of learning, the resulting run-time is geometrically distributed.

This chapter is structured as follows. First, in the introduction we contextualize the work reported in the chapter. Then, section 5.2 is dedicated to perform a run-time analysis of some classical problems in Koza's style GP. This analysis involves some common scenarios, but also some extreme situations in order to understand better the run-time behaviour of the algorithm. Section 5.3 proposes a simple theoretical model of run-time distribution. The main goal of the chapter is addressed in section 5.4, where the need of the run-time analysis is justified, and the results of such analysis are used to propose a model of success probability. The proposed model is then experimentally validated. Some work aligned with ours is presented in section 5.5, followed by a discussion of the results and some final conclusions.

## 5.1 Introduction to run-time analysis

Run-time analysis is a powerful tool used to characterize the run-time behaviour of the algorithms. A common practice in EC is to measure the run-time and report its central tendency and variability statistics. However, this practice has some drawbacks. Perhaps, the most remarkable one is that such a concise reporting necessarily has to drop relevant information. Using a full description of the measured run-times is probably a better practice because no information is lost in the process, and perhaps more importantly, it opens the statistical characterization of the run-time, which can lead to important observations and more solid statistical methods.

To the authors' knowledge, the first use of a run-time analysis was performed by Feo *et al.* in [82]. The most widely used tool in run-time analysis is the *Run-time Distributions* (RTDs), that was introduced, formalized, widely studied and advocated by Hoos and Stützle in [116]. It was followed by an extense sequence of publications where several stochastic search algorithms and problems were studied using RTD analysis. We can briefly define a RTD as the empirical cumulative distribution of the probability of finding a feasible solution at time $t$.

There are several advantages of using RTDs. First, it fully describes run-times from a statistical perspective, since all the statistical properties of the data are contained in the RTD, and thus, statistics such as the median and the standard deviation can be calculated from it. In addition, important properties are easily visualized, such as the size of the tails, rapid or slow decays of the probability of finding a solution, and so on. This data is lost in more conventional reporting practices. RTDs also facilitate visual comparison of algorithms and

can also be used to determine optimum restarts. But probably one of its most interesting, and less used, features, is that it opens the doors to introduce parametric statistics to analyze the run-time of the algorithms. In this way, more robust statistical methods can be used, for instance, to determine whether an algorithm A is able to find a solution in less evaluations than an algorithm B.

Run-time analysis made in the context of Stochastic Local Search (LS) and other metaheuristics with optimal parameter configuration have shown that the exponential (or shifted exponential) distribution has a major role to describe RTDs [205]. Depending on the problem difficulty, and the parameters setting, other distributions might appear, in particular the Weibull distribution in easy problems, which is a generalization of the exponential. This result holds in particular for SLS methods, such as WalkSAT, applied to 3SAT, CSP and TSP problems [115]; and some metaheuristics such as Simulated Annealing (SA), Iterated Local Search (ILS), or Ant Colony Optimization (ACO) [205]. These results suggest that RTDs have common properties across different algorithms and problems.

Nonetheless, to the authors' knowledge, run-time analysis has not been applied to tree-based GP, and thus, there is no evidence supporting that the previous results can be applied to GP. In order to develop a model of success probability in section 5.4, we need to know the run-time behaviour of tree-based algorithms, which can be performed using experimental methods.

## 5.2 Run-time analysis of tree-based Genetic Programming

Generally, experiments dealing with Evolutionary Algorithms (EAs) involve running the algorithm until a certain condition is fulfilled or a resource budget is exhausted. Usually the budget of resources provided to the algorithm is, directly or indirectly, time through a limit on the number of evaluations, or generations, in a generational algorithm. Run-time analysis is based on this time, whatever the unit that was used to measure it, and in particular, run-time analysis deals with the measurement and analysis of the time required to find a solution.

A common tool used for run-time analysis is the RTD. Let us name $rt(i)$ the run-time of the *i*th successful run, and $n$ the number of runs executed in the experiment, then the RTD is defined as the empirical cumulative probability $\hat{P}(rt < t) = \#\{i|rt(i) \leq t\}/n$ [115]. Reader should note that the definition of the RTD assumes that run-time is measured in time, which is an architecture-dependent measure. There are several problems associated to measuring time in this way [11]. For this reason, instead of using time as a unit, run-time analysis is usually performed using an architecture-independent time unit, such as number of evaluations in EAs, or algorithm iterations. In this case, the term *Run-Time Length Distribution* (RLD) is used instead.

A run of a stochastic search algorithm might find or not a solution. In the first case, the run is successful. But on the contrary, if the run does not find the solution, the run-time usually takes a cut-off value. The term run-time refers to both, when clearly they have different interpretations. The cut-off execution time is a parameter well known by the experimenter, and thus it is of little interest. Much more interesting is the time required to find the solution, which is the basis of RTD analysis.

For these reason, in the context of run-time analysis, we prefer avoiding the term run-

time, in favor of the more specific term *run-time to success*. In this way, we explicit the time required by a run to find a solution, not considering runs that were unable to find it. In the following, we use the number of generations as an architecture-independent time measure, so, the time required by a run to find a valid solution will be named *generation-to-success*. Given that the experimental setup uses generations as time unit, we will use run-time to success and generation-to-success interchangeably. Please, note that run-time to success and generation-to-success are not defined for those runs that do not reach a solution.

The definition of RTDs involves the ability of an algorithm to find a solution, but also how many time the algorithm needs to find it. This property makes sense in the original context they were introduced, whose objective is algorithm comparison and restart point determination. However, in our opinion, there are some drawbacks with this approach when the objective is the characterization of the algorithm behaviour. Perhaps, the most notorious one is that it mixes two concepts that answer different questions and, more importantly, they have different statistical properties. The shape of the RTD answers when the solution is found, while its height determines how likely is to find a solution, i.e., its *Success Rate* (SR). These are two different sides of the observed phenomena that in out opinion should not be mixed. Moreover, there is a more practical reason behind this position, SR has a binomial nature, as seen in chapter 4, and thus it has some well known statistical properties [21, 15]. In addition, generally RTD does not satisfy the property $P(\infty) = 1$, which makes it more difficult to guess which distribution fit visually.

An alternative, quite näive, and efective method to report the time required by an algorithm to find a solution is just plotting the histogram of that time. Both methods, RTDs and histograms of the run-time to success are equivalent and, indeed, they can be easily transformed to each other, given that the SR was known. It is relevant since it gives us a base to compare the results obtained using RTD analysis with the ones reported in the literature. In the following, following our own advice, we base the run-time analysis on the histograms of the run-time to success, measured in generations, and report the SR independently.

### 5.2.1 Run-time behaviour of tree-based GP classical problems

Previous RTD analysis [115], mainly in the context of SLS and some classical AI problems, has shown that the RTD of hard problems is exponentially distributed. In this section we try to verify whether this observation is repeated in tree-based GP, or on the contrary the run-time can be described using other distributions. To this extent, we have measured the run-time to success that yields as a result of running the canonical Koza-style GP algorithm applied to some well known problems. The unit used to measure time is the generation, and since each generation involves a constant number of evaluations, the results should be extrapolated if time was measured in evaluations. Additionally, the number of generations is a discrete measure, but it will be approximated using continuous distributions in order to compare the results with the literature more easily. This assumption is in opossition to the discrete-time theoretical model that we introduced in section 5.3.

We first consider some classical problems in GP widely used by the literature, and obtain the empirical distribution of the generation-to-success. These problems belong to four problem classes: the artificial ant, k-multiplexer, even k-parity and linear regression without Ephemeral Random Constants (ERC). Two instances of each binary problem were consid-

Table 5.1: Tableau for the problems under study: Artificial Ant with the Santa Fe trail, 6-multiplexer, 11-multiplexer, even 4-parity, even 5-parity and symbolic regression without ERC.

| Parameter | Artificial ant | 6/11-multiplexer | 4/5-parity | Regression |
|---|---|---|---|---|
| Population | 500 | 500 | 4,000 | 500 |
| Generations | 50 | 50 | 800 | 50 |
| Terminal Set | Left, Right, Move, If-FoodAhead | A0, A1, A2, D0, D1, D2, D3, D4, D5 | D0, D1, D2, D3, D4 | X |
| Function set | Progn2, Progn3, Progn4 | And, Or, Not, If | And, Or, Nand, Nor | Add, Mul, Sub, Div, Sin, Cos, Exp, Log |
| Success predicate | $fitness = 0$ | $fitness = 0$ | $fitness = 0$ | $fitness \leq 0.001$ |
| Initial depth | 5 | 5 | 5 | 5 |
| Max. depth | 17 | 17 | 17 | 17 |
| Selection | Tour. (size=7) | Tour. (size=7) | Tour. (size=7) | Tour. (size=7) |
| Crossover | 0.9 | 0.9 | 0.9 | 0.9 |
| Reproduction | 0.1 | 0.1 | 0.1 | 0.1 |
| Elitism size | 0 | 0 | 0 | 0 |
| Terminals | 0.1 | 0.1 | 0.1 | 0.1 |
| Non terminals | 0.9 | 0.9 | 0.9 | 0.9 |
| Observations | Timesteps=600 Santa Fe trail | | Even parity | No ERC $y = x^4 + x^3 + x^2 + x$ $x \in [-1, 1]$ |

Table 5.2: Estimation of the difficulty to find a solution. It reports the number of runs ($n$), number of successful runs ($k$), estimation of SR $\hat{p}$ and Wilson CI of SR with $\alpha = 0.95$, lower ($L_p$) and upper ($U_p$) values.

| | Artificial ant | 6-Multiplexer | 11-Multiplexer | 4-Parity | 5-Parity | Regression |
|---|---|---|---|---|---|---|
| $n$ | 100,000 | 100,000 | 1,000 | 400 | 5,000 | 100,000 |
| $k$ | 13,168 | 95,629 | 333 | 299 | 305 | 29,462 |
| $\hat{p}$ | 0.132 | 0.956 | 0.333 | 0.747 | 0.061 | 0.295 |
| $L_p$ | 0.1296 | 0.9550 | 0.3045 | 0.7027 | 0.0547 | 0.2918 |
| $U_p$ | 0.1338 | 0.9575 | 0.3628 | 0.7876 | 0.0680 | 0.2975 |

ered; $6$ and $11$ lines were used in the multiplexer, while the parity problem used $4$ and $11$ lines. The trail used in the artificial ant problem was Santa Fe, as described by Koza in [136]. In total six problem instances were used in the experiment. In all the cases the parameters settings and implementations used were the ones found by default in ECJ v18. The only exception is the population size and cut off number of generations, which were changed to tune the algorithm according to the problem difficulty, and the number of timesteps used in the artificial ant, which has increased to $600$. A summary of the settings used in this experiment is shown in table 5.1.

Each one of these problems were run a large number of times ($n$) in order to obtain a sufficient number of successful runs ($k$). Some problem instances were run a huge number of times, $100,000$, because they were reused from previous publications where that number of runs were needed. Other problem instances were run fewer times, enough for the purpose of this study. The number of runs, $n$, was chosen depending on the computational resources needed by the experiment, which is strongly correlated with the population size, and the available computational resources. The number of runs, number of successful runs and an estimation of the SR, and condidence interval of the SR using the method of Wilson with $\alpha = 0.95$ of each of the problem instance is shown in Table 5.2. SR provides a rough estimate of the difficulty of the problem, and, as can be seen in the table, the SR found in the six problem instances ranges from easy problems to difficult ones.

The empirical distribution of the generation-to-success of each problem instance was depicted to overlap with some fitted statistical distributions. In order to estimate the parameters of the distribution, R's function `fitdistr()` was used, which implements a maximum-likelihood method. The exploratory experiments tried to fit data using several distributions, including normal, Poisson, Student's t, and Gamma, however, we found that only a small set of these distributions fit well enough to be considered in the study. So, in the following we only take into account the distributions that fit data better, i.e., lognormal, Weibull and logistic, and additionally the normal distribution is also included in order to ease comparison.

The result of this experiment can be observed in Figure 5.1. The first fact that we observe is that the distribution that better models generation-to-success is the lognormal; it

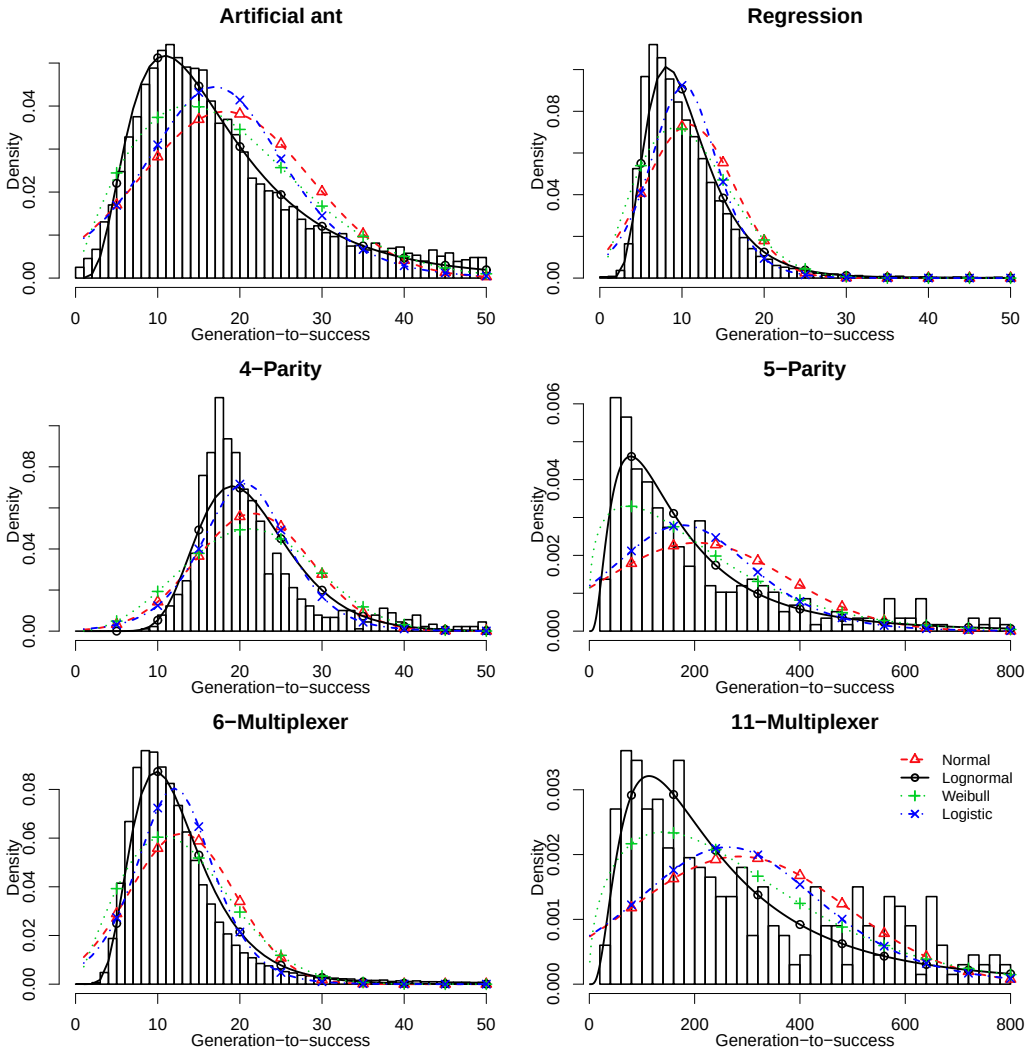


Figure 5.1: Histograms of the generation-to-success of the six problem instances compared to different probability density functions. The parameters of the distributions have been calculated using maximum likelihood estimation. All the available successful runs have been used in the histogram and model fit.

is pretty clear in the case of the artificial ant, which fits nicely the lognormal distribution. The situation is more complex in the rest of the problem instances. In the regression and 6-multiplexer, the lognormal seems to fit well data, however not so well as the artificial ant. In comparison to them, the lognormal fits the generation-to-success worse. All these empirical distributions exhibit a curious fact; in comparison with the lognormal, data shows a pronounced peak, while the lognormal peak is smoother. Additionally, the shape of the histogram decays rapidly after the peak, while the decay in the lognormal is less stepped. The rugosity found in the 5-parity and 11-multiplexer problems might be explained by the lower number of samples used to depict the figures due to the problem difficulty. Finally, the most

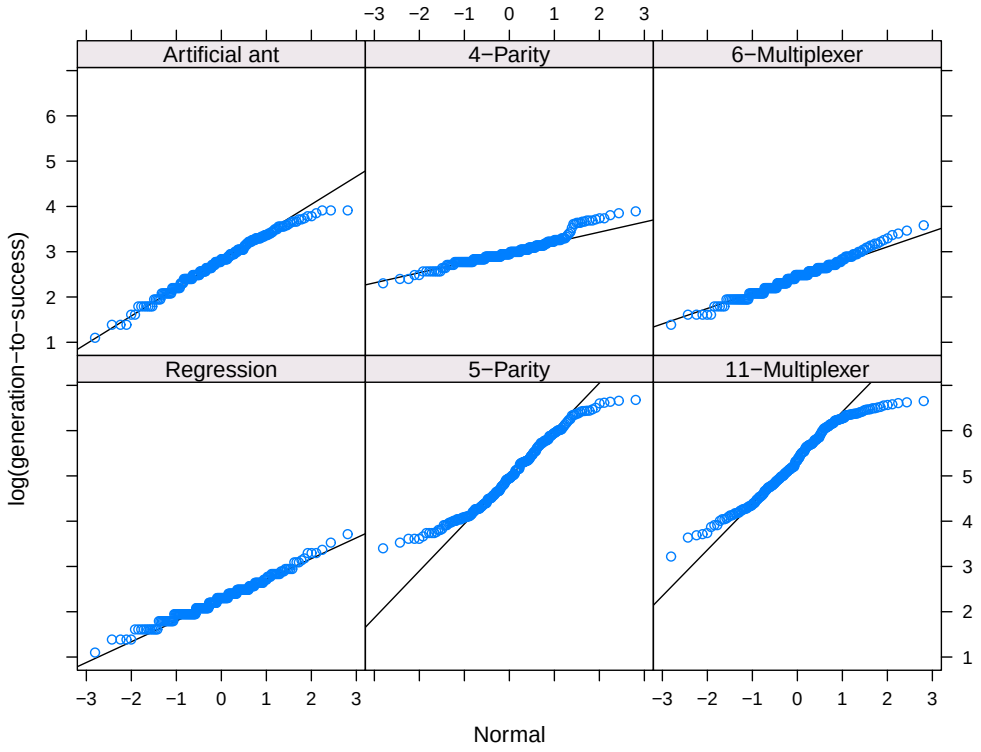


Figure 5.2: Quantile plots of the logarithm of the generation-to-success against samples drawn from a normal population.

erratic behaviour is found in the 11-multiplexer. It has a rough histogram, and is badly fit by any distribution. In any case, the lognormal distribution is the one that seems to fit data better in the six problem instances under consideration.

An important property of the lognormal distribution is its close relationship to the normal one [150]. Normal data might be converted to lognormal data using the exponential function, while, on the other side, lognormal data might be transformed to normal taking natural logarithms. Given this relationship, it seems interesting to represent a comparison between the natural logarithm of the generation-to-success against a normal distribution. This comparison is shown in Figure 5.2, which depicts the quantile plot of the logarithm of the generation to success against a normal distribution. As we could expect, the logarithm of the generation-to-success of difficult problems (5-parity and 11-multiplexer) are not too close from the line that represents the normal distribution; just the opposite than the easy version of these problems, 4-parity and 6-multiplexer. The discrepancy between the peak observed in the data and the lognormal distribution is clearly shown in the two quantile plots of these problems in form of a slight curve on the right of the plot. The logarithm of the generation-to-success of the regression problem also seems to follow a normal distribution, however the tails of the distribution fit worse.

Summarizing the results, lognormal distribution seems to fit reasonably well the generation-

to-success of some of the problems studied, but it fails to describe others. A natural question that rises at this point is why the generation-to-success of some problems follows a lognormal distribution, while others badly fit the lognormal distribution. Looking at the results so far, and related literature, it seems reasonable to hypothesize that the difficulty of the problem, to some extent, influences the distribution, and thus, it seems reasonable to study in more detail the hard problems that did not fit well the lognormal distribution. The next section is devoted to study this question.

### 5.2.2 Run-time behaviour of tree-based GP with difficult problems

It seems that the lognormal distribution plays a major role to describe the generation-to-success in tree-based GP. However, the run-time behaviour observed are only partially described by the lognormal distribution, since some problem instances do not fit well that distribution. So, it seems pertinent to us to study these problems in more detail, in order to gather a more complete perspective about this issue. In particular, we are interested in the 4/5-parity and the 11-multiplexer problems.

Several authors have observed that some SLS algorithms and metaheuristics yield a RTD that follows an exponential distribution [113, 115]. However, histograms in Figure 5.1 showed a shape that are far from the strictly decreasing exponential distribution. Nonetheless, a more detailed observation of the hard boolean problems (5-parity and 11-multiplexer) histograms leads to a more elaborated interpretation. Initially, there is a pronounced increase of the histogram density until the peak is reached, and then it begins to decrease softly. Additionally, the peak of the histogram is more pronounced than all the distributions studied. These observations lead us to hypothesize that the exponential distribution has a role in the picture of generation-to-success distributions in GP.

In order to verify whether there is a hidden exponential distribution, we removed the left tail of all the histograms, and then overlapped a shifted exponential distribution. The shifted exponential is an exponential distribution of the form

$$f(\lambda;t) = \begin{cases} \lambda e^{-\lambda(t-t_0)} & \text{if } t \geq t_0, \\ 0 & \text{if } t < t_0. \end{cases} \tag{5.1}$$

where $\lambda$ is the only parameter of the distribution, and $t_0$ is the shift. The parameter $\lambda$ has been estimated using maximum-likelihood while $t_0$ is the generation where the histogram density reaches its maximum value.

The histogram of the right tail of the generation-to-success, fitted with a shifted exponential, is depicted in Figure 5.3, while the estimated parameters of the lognormal and exponential are shown in the Table 5.3. Those problems that were poorly described by the lognormal distribution (4/5-parity and 11-multiplexer), are reasonably well fit by an exponential distribution when the left tail is removed. Perhaps not surprisingly, the same can be observed in the rest of the problems, so, it seems that removing part of the data eases fit it with the exponential distribution. Nonetheless, there is a notable difference, due to the strong asymmetry of the hard boolean problems, the amount of data that has been removed is much lower in comparison to the rest of the histograms.

Quantile plots comparing generation-to-success and exponential distributions are shown in Figure 5.4. In general, the exponential distribution seems to fit well this subset of the data,

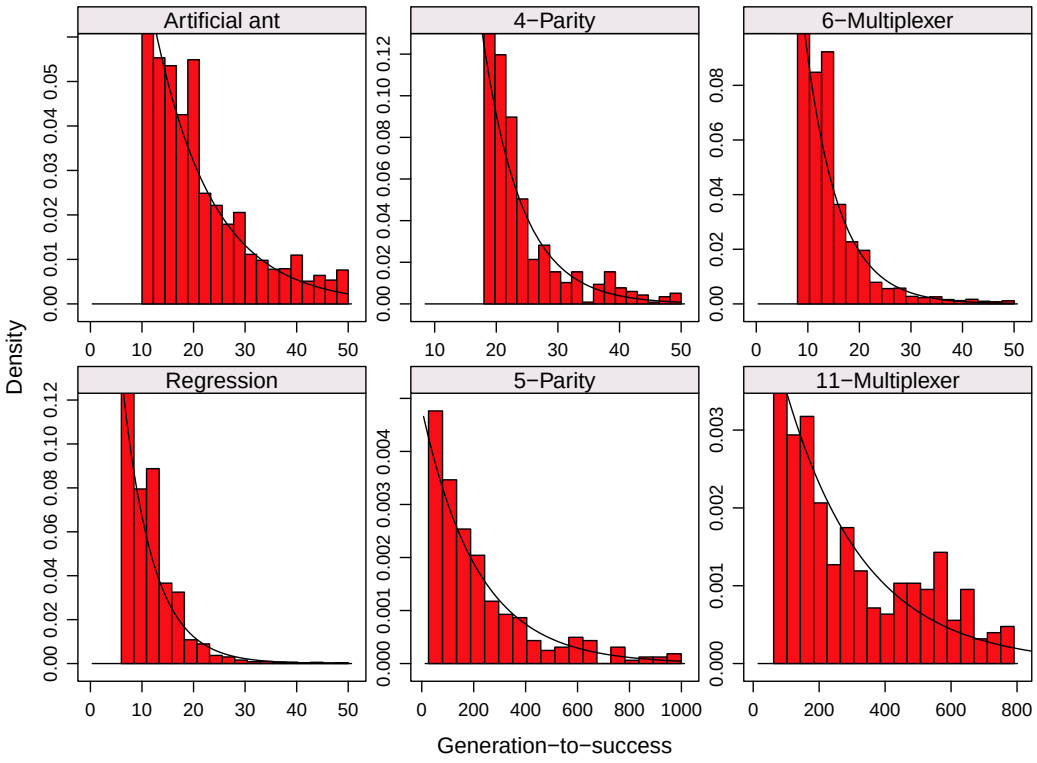


Figure 5.3: Histogram of the right tail of the generation-to-success of the problems under study. An exponential distribution has been overlapped.

with some interesting individual cases. Surprisingly, the right tail of the 5-parity problem instance fits quite well the exponential distribution, when the lognormal distribution failed to model the peak shown in Figure 5.1. In contrast, the right tail of the generation-to-success of the 4-parity problem seems to fit worse an exponential with a tendency to overestimation. It also presents some outliers.

Some authors have observed that the RTD of several algorithms fits well a shifted exponential distribution [113], and explained it hypothesizing the existence of a initialization phase where no solution is found. The plot shown in Figure 5.5 supports that hypothesis, it represents the average tree depth and the average number of nodes. The histogram depicted in Figure 5.1 shows that the density of the generation-to-success begins growing rapidly, and then, after the mode, it decays slowly. This behaviour is more evident in the two hard boolean problems. The maximum in these two problem instances is found arround generation $60$. If we now observe the shape of the average tree depth (Figure 5.5, bottom), we find that initially the average depth also increases rapidly up to a point and then it remains almost constant, in what seems an asymptotic behaviour. Most importantly, that point is placed arround generation $60$ in both problems. This lends credence to the existence of a correlation between the average tree depth and the shift of the exponential distribution that models the right tail of the generation-to-success of boolean hard problems.

Table 5.3: Estimations of the parameters of the lognormal and shifted exponential for the six problem instances under study. The lognormal distribution has two parameters, the mean $\mu$ and the standard deviation $\sigma$, and the shifted exponential has two parameters, $\lambda$ and the shift $t_0$.

| | $\hat{\mu}$ | $\hat{\sigma}$ | $\hat{\lambda}$ | $t_0$ |
|---|---|---|---|---|
| Artificial ant | 2.73 | 0.595 | 0.085 | 9 |
| Regression | 2.289 | 0.44 | 0.150 | 5 |
| 4-Parity | 3.03 | 0.284 | 0.146 | 17 |
| 5-Parity | 5.004 | 0.8 | 0.005 | 7 |
| 6-Multiplexer | 2.464 | 0.425 | 0.149 | 7 |
| 11-Multiplexer | 5.367 | 0.797 | 0.004 | 60 |

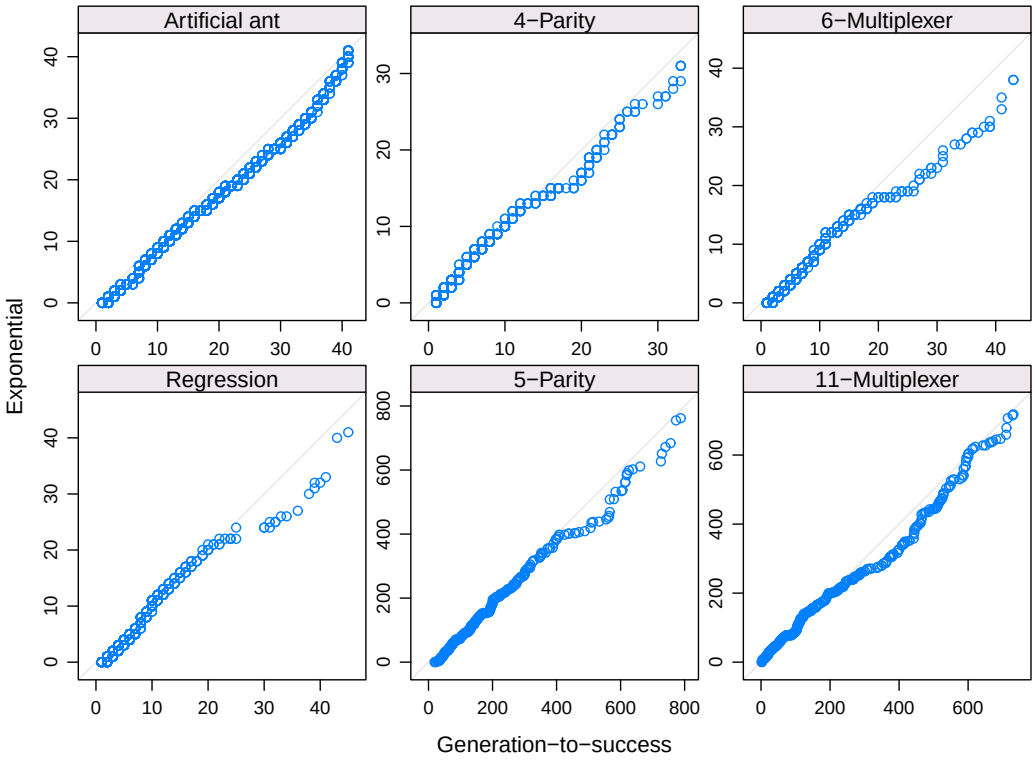


Figure 5.4: Quantile plots comparing the right tail of the generation-to-success and a exponential distribution. Samples higher than the mode have not been included in the plot in order to exclude the initialization phase.

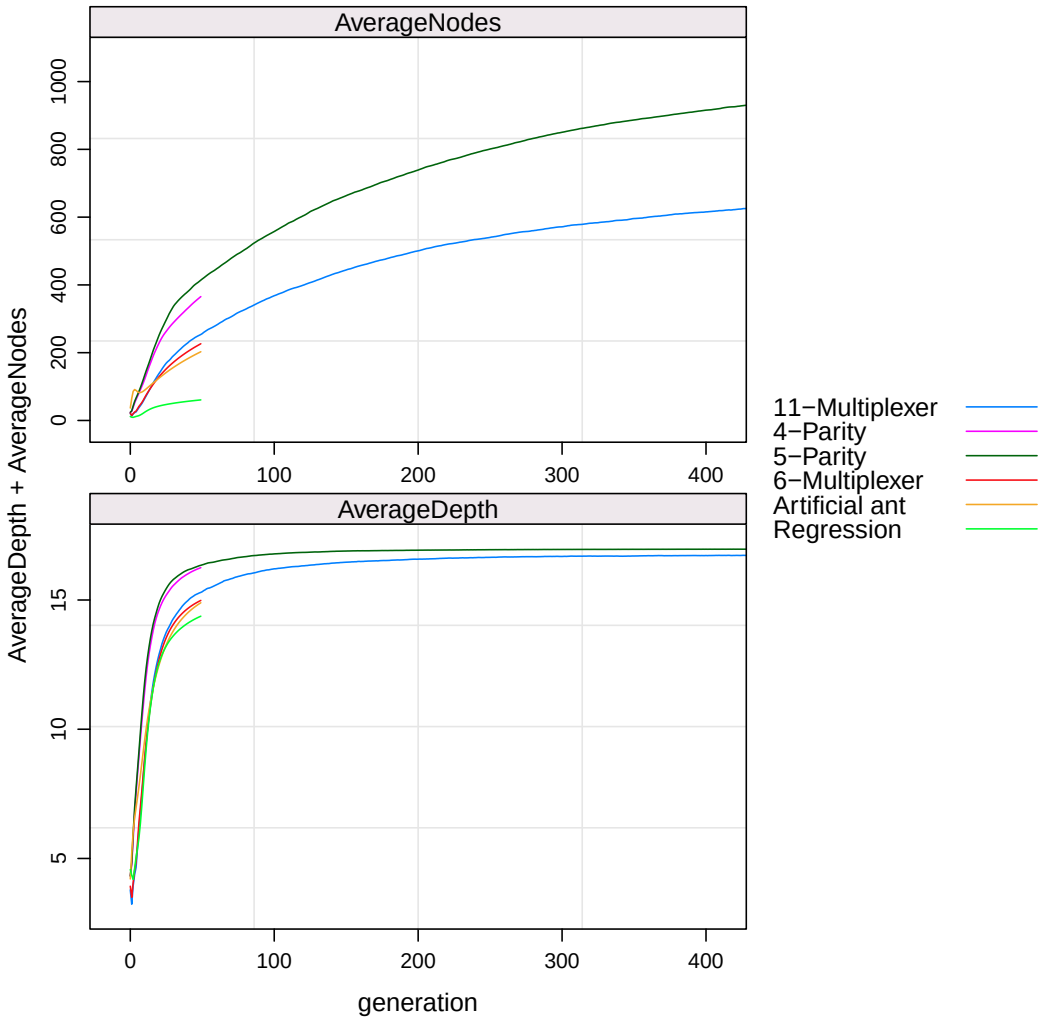


Figure 5.5: Average number of nodes (top) and average tree depth (bottom) of the problem instances under study.

If the correlation between average tree depth and the shift of the exponential distribution were confirmed, it would lead to provide an additional support to the Hypothesis proposed by Hoos, i.e., exponential distribution models the generation-to-success after an initialization phase. During the initialization, trees in average increase their size until the maximum size is reached. Then the search is performed not increasing the tree depth, but its shape, as can be seen in the average number of nodes in Figure 5.5 (top). It is worth to remember that the only bloat control method included in the algorithm used in the experimentation is the Koza style hard limit of the maximum tree depth.

There is an additional argument in favor of the proposed interpretation of the shift found in the exponential, in this case it has more theoretical roots. A key property of the exponential distribution is its lack of memory, because of that it is widely used to model memoryless

processes. An exponential distribution of the generation-to-success suggests that the search is memoryless, but it is clear that the initialization phase of the run has memory since the average tree size increases with time until it stabilizes, which is just the point where data begins to fit the exponential distribution. After that point, trees have reached their maximum depth and the search is less influenciated by the memory.

We should underline that we have considered continuous-time instead of discrete-time. As a consequence, we have not considered the role of discrete distributions, and in particular the role of the geometric distribution, which is interesting because it is the discrete-time counterpart of the exponential distribution and plays a key role in section 5.3. In theory, exponential and geometric distributions should be equivalent, and both should fit equally well (or bad). In order to test it, we fit the geometric distribution in the cases were the exponential distribution fit well the data, finding that they are equivalent, so, the discussion made related to the exponential distribution can also be done with the geometric distribution. This observation is important to relate the results shown in this section to the theoretical discrete-time model that we have proposed in section 5.3.

Experiments carried out in this section suggest that the run-time to success of some difficult problems follow a shifted exponential distribution. The shift of the exponential coincides with the stabilization of the depth growth in the population, suggesting a correlation. So far, we found two distributions, the lognormal and the shifted exponential, that model the generation-to-success of almost all the problem instances studied, with the exception of the 4-parity problem. All the algorithms so far used in the experimentation used a standard parameters setting, yielding the run-time behaviour reported above. A natural question that raises is whether this behaviour can be changed if the parameterization of the algorithm is modified. In order to answer this question, we used an extreme algorithm design.

### 5.2.3 Run-time behaviour of tree-based GP with random selection

The problem instances so far studied have shown a run-time to success that follows a lognormal or a shifted exponential distribution. However, it is unclear if this behaviour is particular of the standard algorithm configuration used so far, or, on the contrary, it is a general property. To study this issue, we run the algorithm with an extreme configuration, and analyze its generation-to-success to verify whether it fits a lognormal or an exponential distribution.

Due to the presence of the exponential distribution previously discussed, it seems reasonable to suppose that randomness in the search is a factor that should be considered. So, in this experiment we increment the randomness of the algorithm by reducing the tournament size. In particular, we have run the same canonical tree-based GP algorithm with the same configuration shown in Table 5.1 with a notable exception, the tournament that originally was set to 7, now has been reduced to 1. In this way, we have removed the selective pressure, making the selection purely at random.

As an initial hypothesis, we can expect that, since exponential distribution deals with memoryless processes, it might describe the generation-to-success of these runs without selective pressure. However, we should avoid a mistake, random selection does not mean it is a memoryless algorithm. Even though the selective pressure has been removed, the algorithm still has memory through the recombination operator, however, the role that it plays and how it might affect the generation-to-success is not clear.

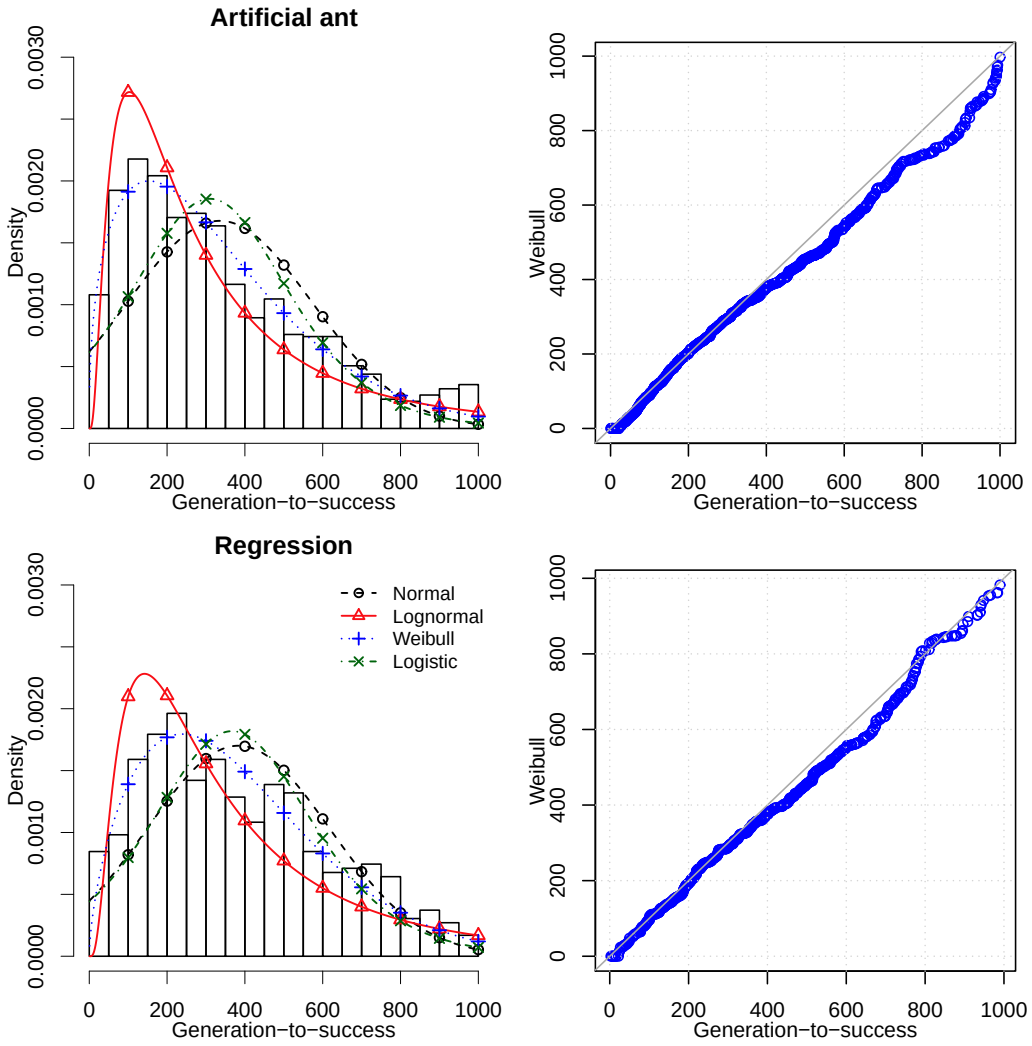


Figure 5.6: Histogram of the generation-to-success of two problems solved by GP without selective pressure compared to four distributions (left) and quantile plots comparing data and samples drawn from a Weibull distribution with its parameters estimated using maximum-likelihood (right).

The histograms of the generation-to-success obtained without selective pressure are shown in Figure 5.6 (left). First we should point out that, since there is no selective pressure, the population is not pushed to any direction, making the search almost random. As a result, the efficiency of the algorithm finding a solution has been reduced notably, and indeed only two out of the six problems instances found enough solutions to be significant. Hence, only two problems are reported in this section, the artificial ant and the regression. In addition, the time required to find the solution has been dramatically increased, those problems that required at most $50$ generations, in absence of selective pressure, require $1,000$ generations to find it, if it is found.

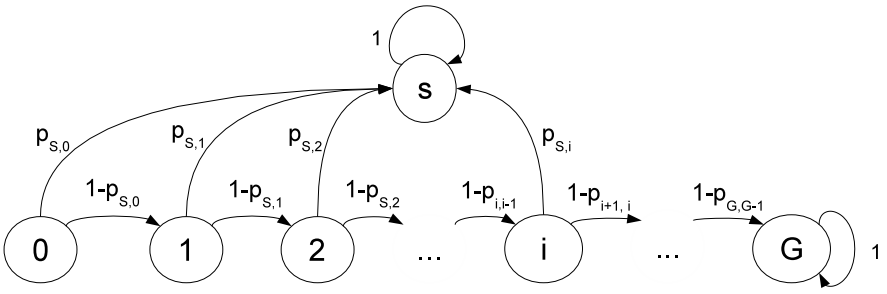


Figure 5.7: Discrete-Time Markov Chain model of an iterative stochastic search algorithm. State $s$ denotes that the algorithm has found a solution to the problem at hand, while states $i \in \{0, ..., G\}$ denotes that no solution was found at iteration $i$.

Perhaps, the most interesting fact in the figure is the shape of the histogram. Surprisingly, the histograms in Figure 5.6 (left) show that in this case either lognormal neither exponential fit data, indeed, a new distribution appears into scene, Weibull, which is able to fit data pretty well. This observation is supported by the quantiles plot depicted in Figure 5.6 (right). It shows that the generation-to-success is well described by the Weibull distribution, even in extreme values. The histograms have peaks somewhat smoother than the lognormal distribution, and a right tail that decreases slower. However, there are some fluctuations in the histograms that degrade the fit, but fortunately they are small.

In summary, in absence of selective pressure the generation-to-success of the observed problem instances fit a Weibull distribution. Curiously, it relates to the previous observation of the exponential nature of generation-to-success in difficult problems because the Weibull distribution is a generalization of the exponential. In any case, the results so far reported are hardly generalizable without a theoretical framework or a much more extense experimental apparatus. The next section is an attempt to provide a theoretical approximation to generalize these results.

## 5.3 A simple theoretical model of generation-to-success

A pertinent question at this point is whether there is a theoretical explanation of the distributions so far found. To try to give an insight to this issue we have modeled the convergence of the algorithm using Discrete-Time Markov Chains (DTMC) [230]. Without lack of generality, we consider a discrete-time model instead of a continuous-time model. In order to generalize the results, we first discuss stochastic search algorithms and then, the problem is restricted to population-based algorithms.

Let $S_n = j$ be a random variable that takes a state $j \in \{0, 1, ..., G, s\}$ at iteration $n \in \{0, ..., G\}$. There are only two feasible transitions from state $j, j < G$, to $j + 1$ or $s$. If the state $G$ had been achieved, then we say that the algorithm has failed in finding a solution to the problem. On the other hand, if after any number of transitions fewer than $G$ the state is $s$, we say that the algorithm has found a solution and thus it has been a success. Once the state $s$ has been reached, the system cannot change its state. The model is better illustrated

in Figure 5.7.

The probability $P(S_n = j|S_{n-1} = i)$ is the transition probability between states $j$ and $i$, for clarity we denote it as $p_{j,i}$. Then, the probability of an algorithm to go from an initial state to a certain final state is given by the multiplication of the transition probabilities found in the path between the initial state and the final state. The transition probabilities are given by the transition matrix $[p]$ as follows

$$[p] = \begin{bmatrix} 0 & 1-p_{s,0} & 0 & 0 & \cdots & 0 & p_{s,0} \\ 0 & 0 & 1-p_{s,1} & 0 & \cdots & 0 & p_{s,1} \\ 0 & 0 & 0 & 1-p_{s,2} & \cdots & 0 & p_{s,2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & 1-p_{G,G-1} & p_{s,G-1} \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{5.2}$$

Such matrix has dimension $(G+2) \times (G+2)$. The column $i$ stands for receiving state $i$, and row $j$ stands for editing state $j$. The last column and row are related to the state of success, $s$. The algorithm always begins in state $0$, so the initial probability vector yields as

$$p(0) = [1, 0, \cdots, 0] \tag{5.3}$$

The (not-accumulated) probability of reaching a succes in exactly $i$ iterations is given by the path $0, 1, .., i-1, s$, which is given by the probability $P(S_i = s|S_{i-1} = i-1, ..., S_0 = 0)$. For convenience, we denote that probability as $p_{s,i-1,...,0}$. With these considerations we can state the following theorem.

**Theorem 1** *The probability of a iterative stochastic search algorithm to find a solution in exactly $i, 1 \leq i \leq G$, iterations is given by*

$$p_{s,i-1,...,0} = p_{s,i-1} \prod_{k=0}^{i-2} (1 - p_{s,k}). \tag{5.4}$$

**Proof 1** *The algorithm can be modeled using the transition matrix shown in (5.2). Then, the probability of finding a solution after $i$ iterations is the probability of reaching to the state $s$ by using the path $0, 1, ..., i-1, s$. Hence, given that they are independent, the probability of the algorithm to follow this path is given by the multiplication of their transition probabilities. So we can infer that*

$$\begin{aligned} p_{s,i-1,...,0} &= p_{s,i-1}\ p_{i-1,i-2}\ \cdots\ p_{1,0} \\ &= p_{s,i-1}\ (1-p_{s,i-2})\ \ldots\ (1-p_{s,0}) \\ &= p_{s,i-1}\ \prod_{k=0}^{i-2} (1-p_{s,k}) \end{aligned} \tag{5.5}$$

**Corollary 1** *The run-time of a memoryless stochastic search algorithm to find a solution, measured in iterations, follows a geometric distribution.*

**Proof 2** *Given a memoryless algorithm, the transition probabilities are equal,* $p_{i,i-1} = p_{j,j-1}\ \forall i,j \in \{1,\cdots,G\}$. *Taking* $p_{s,i} = p$ *in (5.4) yields*

$$p_{s,i-1,\ldots,0} = p\prod_{k=0}^{i-2}(1-p) = p(1-p)^{i-1}, \tag{5.6}$$

*which is the geometric probability mass function.*

Theorem 1 supposes an iterative search algorithm, however, when dealing with population-based algorithms, as is the case in EAs and GP, there is a population of candidate solutions, so, to be more realistic in the context of EAs, they should be considered. Then, the transition probability $p_{a,b}$ can be expressed as a function of the success probability of each individual in the population. Let us name it $p_{a,b}^{(k)}$, where $(k)$ stands for the probability of individual $k$ to find a solution, where $k \in 1,...,M$ and $M \in \mathcal{N}^+$ is the number of individuals in the population.

With these considerations, the probabilities $p_{i+1,i}$ and $p_{s,i}$ can be decomposed as a function of the success probability of each individual in the population. The probability $p_{i+1,i}$ is the probability of not finding any solution by any individual in the population, while $p_{s,i}$ is the inverse of $p_{i+1,i}$. This idea can be expressed analytically by

$$p_{i+1,i} = \prod_{k=1}^{M}(1-p_{s,i}^{(k)}) \tag{5.7}$$

$$p_{s,i} = 1 - p_{i+1,i} = 1 - \prod_{k=1}^{M}(1-p_{s,i}^{(k)}) \tag{5.8}$$

With these considerations we can particularize theorem 1 to generational population-based algorithms, in particular to EAs. In order to be consistent with the experimentation, we consider that each generation is an algorithm iteration.

**Theorem 2** *Given a generational EA with a population size* $M$*, the probability of finding a solution at a certain generation* $i$ *is described by*

$$p_{s,i,1-1,\ldots,0} = \left(1 - \prod_{k=1}^{M}(1-p_{s,i}^{(k)})\right)\prod_{j=0}^{i-2}\prod_{k=1}^{M}(1-p_{s,j}^{(k)}) \tag{5.9}$$

**Proof 3** *Equation (5.9) comes from using (5.7) and (5.8) with (5.4) and some basic algebraic manipulation.*

**Corollary 2** *The run-time of a memoryless generational EA, measured in generations, is described by a geometric distribution.*

**Proof 4** *In a memoryless algorithm, the probability of an individual to find a solution is independent of the generation, and therefore the transition probabilities are equal,* $P^{(k)}(i|i+1) = P^{(k)}(i+1|i) = p^{(k)'}$. *Subsequently, (5.9) can be simplified,*

$$p_{s,i,i-1,\ldots,0} = \left[1 - (1-p^{(k)'})^M\right]\prod_{j=0}^{i-2}\left(1-p^{(k)'}\right)^M \tag{5.10}$$

*This equation can be simplified with a change of variable* $(1-p^{(k)'})^M = (1-p')$*, then*

$$p_{s,i,i-1,\ldots,0} = p' \prod_{j=0}^{i-2}(1-p') = p'(1-p')^{i-1}, \tag{5.11}$$

*which is the geometric probability mass function. An alternative proof of this theorem is setting* $M = 1$ *in (5.9), yielding to (5.4).*

Experiments carried out showed three distribution involved in the description of the generation-to-success: lognormal, Weibull, and exponential. Due to the direct relationship between exponential and geometric distribution, corollaries 1 and 2 provide a theoretical framework to describe the exponential behaviour found in some problem instances, i.e., the absence of memory in the search process. The theoretical explanation of the appearance of the Weibull and lognormal distributions is unknown. Equation (5.9) might provide a clue about it. It shows that the probability $p_{s,i-1,\ldots,0}$ can be expressed as the multiplication of several terms, and thus, it would make sense to apply the Multiplicative Central Limit Theorem to conclude that $i$ is a lognormal random variable. Nonetheless, this interpretation would be naïve since it supposes that the distribution is always lognormal, which contradicts the empirical data previously reported.

In this section we have proposed a theoretical model of the probability density of the generation-to-success based on DTMC. It was also shown that when success probability in each generation is constant, and therefore the algorithm is memoryless, the underlying distribution of the generation-to-success is a geometric one. When a lognormal or Weibull distribution models the generation-to-success is an open issue. Nonetheless, related work in other areas related to stochastic search might provide clues, and additional evidence about the generality of these results. Once the run-time behaviour of tree-based algorithms has been characterized, we are in position to propose a model of success probability.

## 5.4 A new model of success probability

In order to develop a model of success probability, it is convenient to define more formally some terms, in particular the terms success probability and success rate. In EC, an experiment uses to be composed by a collection of $n$ independent runs. Due to the random nature of EAs, many of their properties are stochastical, and thus they cannot be characterized using a single run, but with an experiment. One of these properties is the *success probability*, or, using Koza's notation [136], $P(M,i)$, where $M$ is the population size, and $i$ the generation number.

$P(M,i)$ is calculated as the ratio between the number of successful runs in generation $i$, $k(M,i)$, and the number of runs $n$ in the experiment,

$$P(M,i) = \frac{k(M,i)}{n} \tag{5.12}$$

This estimator of the success probability is also its maximum likelihood estimator [174].

We define the SR as the accumulated success probability in an infinite number of generations, so $SR = \lim_{i\to\infty} P(M,i)$. The reader would agree with us if we state that running the algorithm for an infinite number of generations is not a general practice. Usually

an experiment is run for a fixed finite number of generations, $G$, then the SR is given by $SR = \lim_{i\to\infty} P(M,i) \approx P(M,G)$. Since the true SR can hardly be measured in experiments, $P(M,G)$ is just an approximation to SR, and thus it can be seen from a statistical perspective as an estimator $\widehat{SR} = P(M,G)$.

### 5.4.1 A general model of success probability

Let us consider the problem from another point of view. Instead of analyzing the problem looking at the probability $P(M,i)$, we pay attention to the generation-to-success. May $x_i^k$ be the outcome of run $k$ at generation $i$, and can take two values labeled as "$success$" and "$failure$", so $x_i^k \in \{\text{"}success\text{"}, \text{"}failure\text{"}\}$ for $k = 1, 2, \ldots, n$ and $i = 1, 2, \ldots, \infty$, therefore we assume an EA that is run an infinite number of generations. We suppose that if $x_i^k =$ "$success$" then the outcome in the next generation is also success, $x_{i+1}^k =$ "$success$". Using this probabilistic notation, $P(M,i)$ can be expressed as $P(x_i^k =$ "$success$"$)$. In order to simplify the notation, in the following, labels "$success$" and "$failure$" will be equivalent to "$s$" and "$f$".

Now consider the success generation $g^k$ in which run $k$ converges to the solution, so $g^k \in \mathbb{N}^+$ is a random variable that may take any positive integer value. The probability of run $k$ to have converged in generation $i$ is $P(g^k \leq i)$, while the success generation is not defined for those runs that have failed. Similarly, the SR of an algorithm might be expressed as $SR = P(x_\infty^k = s)$.

Our objective is to find a model of $P(M,i)$. This model, using the probabilistic notation defined before, and from the perspective of success generation, is

$$P(M,i) = P(g^k \leq i)$$

This equation does not take into account the total number of successful runs, so it not useful from a practical point of view, it is necessary to express this probability as a function of something that could be measured. To consider this fact, we can use a conditional probability. Then, a run has a success generation if, and only if, its outcome has been a success, so

$$P(g^k \leq i | x_\infty^k = s) = F(i) = \int_0^i f(u)du \tag{5.13}$$

where $f(u)$ is the (yet unknown) probability density function (PDF) of the random variable $g^k$, while $F(i) = \int_0^i f(u)du$ is itscumulative density function (CDF).

We are now in condition to use Bayes' Theorem,

$$P(g^k \leq i | x_\infty^k = s) = \frac{P(g^k \leq i)P(x_\infty^k = s | g^k \leq i)}{P(x_\infty^k = s)}$$

Obviously, the conditional probability $P(x_\infty^k | g^k \leq i)$ equals 1 because if the experiment has converged into a solution, by definition, its outcome has been "$success$". $P(g^k \leq i | x_\infty^k = s)$ is not known, but can be empirically studied, and $P(x_\infty^k = s)$ is the SR, which also can be easily estimated. Solving the equation for $P(g^k \leq i)$ is straightforward to conclude that

$$P(g^k \leq i) = P(x_\infty^k = s)P(g^k \leq i | x_\infty^k = s)$$

following that

$$P(M,i) = SR\ F(i) \tag{5.14}$$

It provides an alternative representation of the accumulated success probability, more convenient for this study, furthermore, it is a generalization of the binomial random variable described in chapter 4. If assume $i = \infty$, then the CDF $F(\infty)$ equals 1, and thus $P(M,\infty) = SR$, which is binomial. So this model is composed by two terms, the fractional term that describes the SR, while the second term models the variation in time of that probability.

If we fix the generation to an arbitrary value $i_0 < \infty$, the underlying distribution is again binomial, as we could expect. However it is multiplied by a factor $F(i)$ that depends on the generation $i$, and represents the proportion of runs that are supposed to have found a solution in generation $i$.

The model shown in (5.14) presents a serious problem, it has been constructed under the assumption of an experiment that has been run an infinite number of generations. We guess that the reader will agree with us if we state that this situation is rather difficult to happen in real life. It is impossible to measure SR, but instead we can estimate it as $\widehat{SR} = k(M,G)/n$, with $G < \infty$. Of course, this measure is unlikely to be equal than SR, so, we will be introducing an error that is specific of this model. So, the model given by (5.14), cannot be used in practice, but instead

$$P(M,i) = \widehat{SR}\ F(i) = \frac{k(M,G)}{n} F(i) \tag{5.15}$$

Unfortunately, the model given by (5.15) does not provide a close form of $P(M,i)$ and thus it cannot be used. However, we only need a model of $P(M,i)$ to make (5.15) useful. But this problem was, to some extent, solved in the run-time analysis performed in section 5.2. It was showed that in tree-based GP, the lognormal distribution seems to fit well the generation-to-success, and it is reasonable to suppose that generation-to-success is a lognormal distribution. With this data, we can reformulate (5.15) to provide a complete model of success probability.

### 5.4.2 A specific model of success probability

Experimentation reported above showed that the lognormal distribution fits reasonable well most of the case studies, in particular, all with the exception of the two hard boolean problems. So it seems reasonable to assume a lognormal distribution from this point. If we make this assumption, then it is straightforward to then deduce a model of $P(M,i)$ from (5.14) that could be used in practice.

It is well known that the lognormal CDF [150] is given by

$$F(i;\hat{\mu},\hat{\sigma}) = \Phi\left(\frac{\ln i - \hat{\mu}}{\hat{\sigma}}\right) \tag{5.16}$$

where $\Phi(...)$ is the standard normal CDF. If there are $m$ runs that have converged in the experiment, and $g_k, k = 1, ..., m$ is the generation-to-success of run $k$, then

$$\hat{\mu} = \frac{\sum_{k=1}^{m} \ln g_k}{m} \tag{5.17}$$

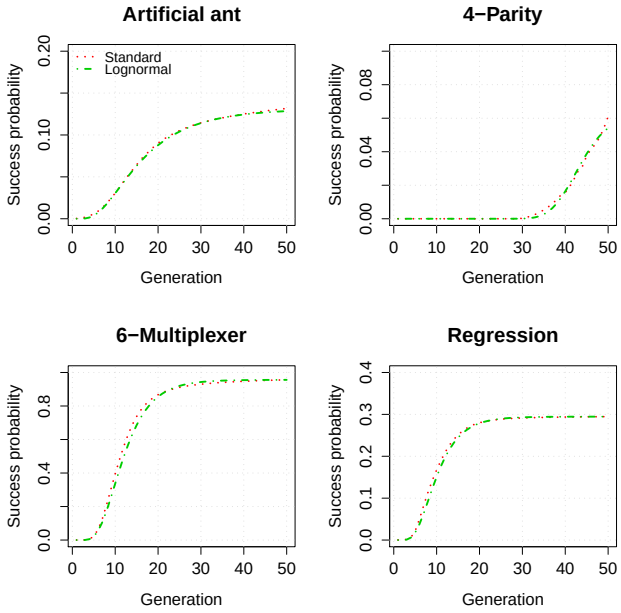


Figure 5.8: Comparison among the best maximum-likelihood estimator of the accumulated success probability and the model approached using a lognormal distribution.

and

$$\hat{\sigma} = \sqrt{\frac{\sum_{k=1}^{m} (\ln g_k - \hat{\mu})^2}{m}} \tag{5.18}$$

Using (5.14) and (5.16) yields that the accumulated success probability can be expressed as

$$P(M, i) = \frac{k(M, G)}{n} \Phi \left( \frac{\ln i - \hat{\mu}}{\hat{\sigma}} \right) \tag{5.19}$$

All the parameters involved in this equation are known by the experimenter. One advantage of using a lognormal distribution is its close relationship with the normal distribution, and actually, lognormally data can be transoformed into normal data, and viceversa [150], therefore the statistical proporties of the estimator $\hat{\mu}$ and $\hat{\sigma}$ are well known.

### 5.4.3 Experimental validation of the model

Although data has been fitted a lognormal distribution, there is no experimental support to claim that the model of accumulated success probability given by (5.19) is a correct model. So, additional experimental evidence is collected in this section.

Figure 5.8 shows a comparison between $P(M, i)$ calculated using the standard maximum-likelihood method and the lognormal approximation. All the samples available in the datasets were used to calculate $P(M, i)$ with both methods. It can be seen in Figure 5.8 that both methods achieve very similar results, and thus, in the study cases under consideration, when using a large number of runs, our proposal achieves estimations of $P(M, i)$ pretty close to the standard method. Nevertheless, this experiment shows an unrealistic scenario since the

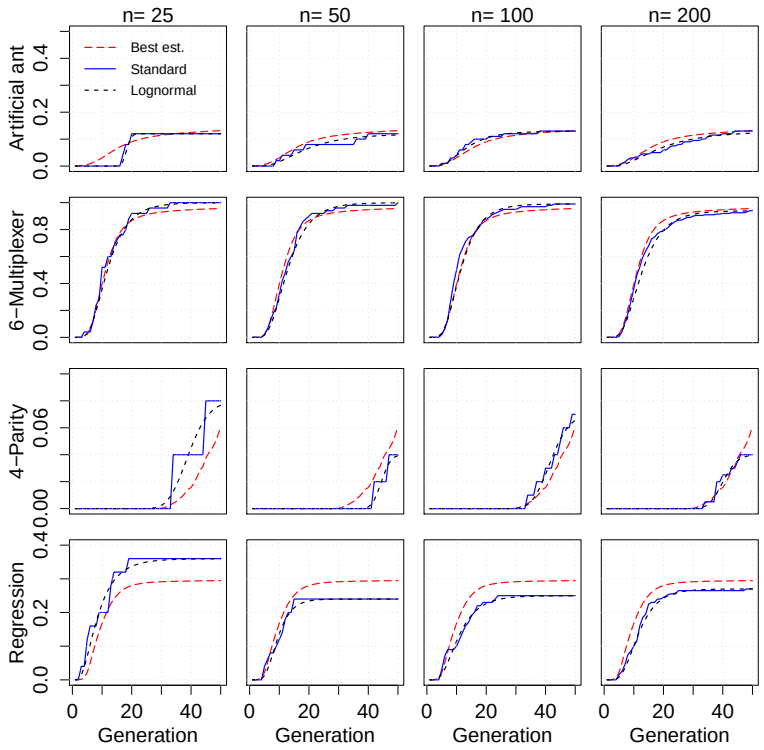


Figure 5.9: Comparison between the best maximum-likelihood estimator of the accumulated success probability and the model approached using a lognormal distribution.

computational cost of running an experiment with real-world problems imposes a maximum number of runs much lower than the used in this experiment.

A collection of experiments were simulated using different values of $n$. Given the whole set of runs stored in the previous experiments, $25$, $50$, $100$ and $200$ runs were resampled with replacement, $P(M, i)$ calculated using both methods and finally they were depicted in Figure 5.9. To give more elements to compare with, the best estimation of $P(M, i)$ (calculated with all the runs) also included in Figure 5.9.

As can be seen in Figure 5.9, when there are a high number of runs, the differences between the three curves tend to disappear, and the estimation with both methods tend to be closer to the best estimation available. More interesting is the relationship between the two methods, they yield similar estimations of the accumulated success probability, which is logical because they use the same data; if one method makes a bad estimation of the accumulated success probability, the other one also makes a bad estimation. It leads us to an almost tautological conclusion: there is no magic. With a small number of runs, there is not much information available, and without information, it is simply not possible to reach good conclusions.

Despite the lack of magic of the proposed method, Figure 5.9 shows an interesting property: the proposed method is able to interpolate values using the experimental data, yielding much smoother curves than the standard method. And apparently, it can be done without

sacrificing the accuracy of the measure. This fact is rather clear, for instance, in the 4-parity problem with $n = 25$. A similar property is the ability of the lognormal approximation to extrapolate values of the accumulated success probability. This is interesting in early generations, where there are no success due to a low, but not null, success probability. In these cases the standard method yields a null estimation of the success probability while the lognormal yields a non zero value.

Another interesting fact that can be found in Figure 5.9 is the excellent estimation made in the 4-parity problem. Despite the fact that the experiment was run for too few generations and it was the domain with the poorest model fit, it generates a nice approximation to the maximum-likelihood estimator of the accumulated success probability. This fact could be quite interesting to reduce the number of generations needed to study the performance of GP using less computational resources, however we feel that this issue requires more study.

## 5.5 Run-time analysis in other Metaheuristics

Experimentation reported in this chapter has been based on Koza's style GP. The experimental results obtained so far cannot, without additional considerations, be generalized to other algorithms. Fortunately, there are some papers describing the run-time behaviour of several SLS algorithms and metaheuristics applied to solve different problems. A comparison of the results reported in the literature and our results might provide some clues about whether the observed distributions are generalizable or not.

Probably the authors that have investigated in more detail this topic are Hoos and Stützle, who applied RTD analysis to different algorithms and problems. They studied in [116] the RLD of WSAT algorithms used to solve 3SAT problem instances. They found that the RLD is exponential when the parameters setting is optimal, shifted exponential or Weibull when the parameters setting is not optimal. Shifted exponential appears when the parameters are above optimal, and Weibull, when they are under the optimum value. Analogously, in [113] Hoos and Stützle studied the RLD of some other stochastic SLS algorithms, such as GWSAT, GSAT with tabu-lists, TMCH and WMCH, to solve instances of SAT and CSP, finding that, again, when parameters are optimal, the RLD follows an exponential, and otherwise RLD fits a Weibull distribution. Curiously, this result only holds for hard instances, in easy instances they did not found statistical significance. RLD of easy instances deviates from the exponential distribution, and they conjectured that it was caused by the initial hill-climb phase of the search. In a later work [114] observed that the RLD of hard 3SAT instances solved with the WalkSAT algorithm also follows an exponential distribution, and more interestingly, the higher the difficulty of the problem, the higher the fit is found. Stützle and Hoos also studied the RTD of ILS algorithms in various types of TSP problems, finding that the RTD follows a shifted exponential distribution [224].

A more recent work made by Hoos and Stützle showed a more complete view about this topic [115]. The authors compare various versions of GSAT and WalkSAT algorithms to solve some problems coded as 3SAT (random 3-SAT, graph coloring, block world and logistic planning) using an RTD analysis. They found that these algorithms also have exponential RTDs for hard problems and high values of the noise parameter. More importantly, they found that RLDs of easy problems are not exponential, despite their tails are still expo-

nential. Their interpretation is that there is an initial search phase where the probability of finding a solution is very low. In hard problems where the run-time is high enough, this initialization phase has a minimal impact, while in easy problems it takes a notable amount of time and thus the distribution is less likely to be exponential. The initial search phase seems to be well described by the Weibull distribution. Only few cases were not correctly described by either exponential neither the Weibull distributions, but even in this case, they can be well approximated by a weighted linear combination of Weibull distributions. Another interesting result reported in [115] is how the noise parameter affects the RLD. For values of the noise higher that optimal, the RTD is still exponential, moreover, the initial search phase is less prominent, increasing the fit of the model. On the contrary, for suboptimal values of the noise parameter, the RLDs exhibits longer and heavier tails compared to the exponential distribution.

Chiarandini and Stützle [53] studied the RTD of ILS, ACO, Random Restart Local Search and two variants of SA applied to the Course Timetabling problem, finding that the Weibull distribution approximates well the RTDs. Data provided in that paper does not permit to clearly deduce whether the approximated Weibull distribution can be reduced to an exponential distribution in any of the problem instances. They report, however, that in SA, the RTD in hard problems can be approximated, at least partially, using a shifted exponential distribution. This result partially contradicts the one reported in [224], where the RTD follows a shifted exponential, but this discrepancy could by explained by the parameters setting and the problem difficulty. In any case, both studies are consequent with the basic fact that RTDs appear to follow a exponential or a Weibull distribution.

On the contrary than Hoos, Stützle and Chiarandini, Frost *et al.* [89] studied the RTD using the same algorithm, backtracking, with different problem instances of the CSP, and found an interesting fact. The RTD of the algorithm running on solvable instances [80] follows a Weibull distribution, while unsolvable instances generate lognormal run-time distributions. However, only the lognormal distribution for solvable problems had statistical significance. These results hold for several backtracking algorithms, and results with the 3SAT problem are similar. It is interesting to mention that, although Frost *et al.* did not mention it, some of the experimental results reported in [89] suggest also an exponential distribution, which is supported by the reported fit $\beta \approx 1$. Other studies about the RTD of several metaheuristics have observed that their RTD follow an exponential or shifted exponential distributions, these metaheuristics include GA, TS and GRASP [205].

It is worth to compare our results with the related work. The main role found for the exponential distribution reported by the literature was not found in our research, which points to the lognormal distribution and reserves the exponential for some difficult problems after some data filtering. However, we have to take into account that we did not looked for optimal parameters, that might have an impact in the result, as suggested in [113, 115].

The interpretation of the Weibull distribution made by Hoos and Stützle in [115] might be applied here. They suggested that the Weibull distribution in hard problems is able to model the initial search phase. The asymptotically exponential behaviour of the Weibull distribution supports this idea. In our case, runtime-to success of hard boolean problems approximate shifted exponential, with a, in comparison, small initial search phase. The Weibull distribution appeared when the selective pressure were removed, which is clearly suboptimal, and thus the search was almost random. More research should be done to determine the influence

of the parameters setting on the run-time behaviour in GP. In case there were a correlation between that and the RTD, it would be a result with a strong practical interest.

## 5.6 Discussion

There are some interesting issues that arise from the observations made in this chapter and the related work. It seems clear, looking at the related work, that RTD analysis is dominated by the Weibull and the exponential distributions. The lognormal, in some circumstances, also appears in the literature. The prominent role of these distributions does not seem to be circumscribed to a particular algorithm, but it seems instead to be a general property of metaheuristics. In addition, this work has shown, with some support from the literature, that the lognormal distribution also plays a role, at least, in tree-based GP.

These three distributions are widely used in Reliability Theory to model failures in physical systems, suggesting a link between run-time analysis and reliability theory. This observation is not new, previously Hoos [115], Luke [156] and Frost [89] mentioned this idea. More recently, Gagliolo and Legrand introduced a detailed discussion about this topic in [90].

An insight to a possible reason behind the run-time to success behaviour found so far was suggested by Frost *et al.* [89], inspired by the Reliability Theory. The failure rate is defined in Reliability Theory as $h(t) = f(t)/(1 - F(t))$, where $f(t)$ and $F(t)$ are, respectively, the density function and CDF. In the context of EAs, given a algorithm that has not found a solution at time $t$, $h(t)+\Delta t$ is the probability of solving the problem in the interval $(t, t+\Delta t)$. In Reliability Theory, $h(t)$ determines when failures happen. If it is constant, the time to failure is exponentially distributed. If $h(t)$ is of the linear form $h(t) = \lambda^{\beta}\beta t^{\beta-1}$ time to failure follows a Weibull distribution. Finally, in case that the failure rate were nonmonotone the lognormal distribution appears. So, this interpretation suggests that the distribution of the generation-to-success depends on the success probability at each generation. Ironically, failures in Reliability Theory can be associated to the run-time to success in GP.

It has been shown that, at least in one extreme case, the parameters settings may change the distribution of the run-time to success. When the default ECJ parameters setting were used, the run-time to success of four out of six problem instances, measured in generations, was characterized by a lognormal distribution, meanwhile if the tournament size was reduced to one, the distribution changed to a Weibull. It opens the question of whether the opposite deduction could be done, given the distribution of the run-time to success, infer facts about the parameter configuration, for instance, its goodness. In our opinion, the practical consequences that such analysis might have, deserves futher research. Moreover, there should be a theoretical reason that explains why the run-time to success is lognormal, exponential or Weibull. Experimentation has provided some clues for such theory, problem difficulty and selective pressure are probably factors to take into account.

The presence of exponential distributions leads to an observation with potential theoretical implications. The exponential distribution is well known for one remarkable property, it is memoryless. This property has been widely used by the literature to support the idea that when the RTD is exponential, it cannot take benefit from restarting the run [116]. We conjecture that there are notable theoretical implications behind this fact. Hoos and Stïzle [115] suggested an interpretation of the exponential nature of RTDs when they stated that “the

exponential RTDs suggest a novel interpretation of Stochastic Local Search behaviour as independent random picking from a drastically reduced search space", this observation might well be valid in GP. Following Hoos' reasoning, we might guess that, given an exponentially distributed run-time to success, the search is memoryless, and hence learning can hardly be.

The lognormality of the run-time to success opens some interesting applications. Just mention one, we could apply it to determine objectively the run-time cutoff value. The determination of the number of runs needed to estimate the parameters is a classical problem in statistical inference, and the transformation between normal and lognormal is straightforward, $X \sim N(\mu, \sigma) \Rightarrow e^X \sim LN(\mu, \sigma)$ and $X \sim LN(\mu, \sigma) \Rightarrow \ln(X) \sim N(\mu, \sigma)$ [150], using this transformation the number of runs in the exploratory experiment can be determined using

$$n = \frac{z_{\alpha/2}^2 \; s^2}{e^2} \tag{5.20}$$

where $e$ is the desired level of precision for the estimation of the mean, given in the same unit than $s$, $s$ the standard error of the samples and $z_{\alpha/2}$ is the upper-$\alpha/2$ critical point from $N(0,1)$ [218]. The probability of getting at least one success in generation $i$ is given by $P(M,G)\ F(i;\hat{\mu},\hat{\sigma})$, while the probability of not getting any success in generation $G$ is

$$\varepsilon = 1 - P(M,G)\ F(G;\hat{\mu},\hat{\sigma}) \tag{5.21}$$

This equation provides an estimation of the error $\varepsilon$ that is introduced by limiting the maximum number of generations to $G$. Moreover, if we set a maximum error that is tolerable, we could calculate $G$ from (5.21), yielding the maximum number of generations that the algorithm should be executed to achieve that error.

## 5.7 Conclusions

The main goal of this chapter was to develop an analytical model of the success probability. In order to accomplish this goal, we needed first to obtain an statistical characterization of the time required by GP to find a solution. For this reason we empirically studied the run-time to success of six well known problems in tree-based GP. As unexpected side effect, we observed some patterns in the run-time behaviour of the algorithms that could lead to general conclusions with an impact wider than the original object of study that motivated this work.

We have found that when using the ECJ default parameters settings, the generation-to-success tends to follow a lognormal distribution. In some cases the lognormal distribution yields smoother peaks in comparison to empirical data, but in general it fits well, and better in any case than other distributions such as the Weibull. There were, however, two problem instances, both difficult boolean problems, whose generation-to-success were not well modeled by the lognormal, but instead by a shifted exponential distribution. A third problem instance, the 4-parity, was not well modeled by any distribution under consideration in this study. We conjecture that there is a initialization phase, to some extent related to the tree size, that is well modeled by lognormal distribution in easy problems, but in hard problems the long run-times make the initialization phase less influential, and the lognormal is no longer a good model. If the initialization phase is removed, the remaining samples fit the shifted

exponential. Finally, in absence of selective pressure, it seems that the generation-to-success does not follow either a lognormal neither an exponential, but a Weibull distribution.

In an attempt to provide some clues to develop a theoretical explanation of the run-time behaviour found in the experiments, we developed a simple theoretical model based on Discrete-Time Markov Chains, and demonstrated that in absence of memory, the generation-to-success follows a geometric distribution. Empirical data also fit well geometrical distribution. In a near future we expect to develop this model and use Montecarlo simulation to understand in which circumstances the lognormal and Weibull distributions appear. More experimentation could be used to provide clues to direct development of a run-time theory. In any case, experiments shown in this chapter, and experiments reported in previous literature have shown a complex picture, where several statistical distributions are involved and there are complex iterations with the parameters settings.

In relation to the dissertation main research goal, the main contribution of this chapter is the proposal of a model of success probability in generational tree-based GP. This model considers the existence of two different -although related- problems: whether the algorithm is able to find a solution, and, given that it has been found, when it happens. The model uses the result of the run-time analysis performed, in particular the fact that the lognormal distribution seems to describe well the run-time to success of most of the studied problems. If the generation is fixed, a classical binomial distribution is derived from our model. Following it, we discussed some practical applications of the model. For instance, given that the generation where the algorithm finds a solution (i.e. the *generation-to-success*) could be described with a known probability distribution, it would be determined when the algorithm is more likely to find a solution, and therefore, use this information to set the maximum number of generations in a well grounded way.

Once that the main component of Koza's computational effort has been analitically modeled, and that the estimation error of SR has been characterized, we can join these two results in order to characterize the reliability of the computational effort. This task is performed in the next chapter.

# Chapter 6

# Accuracy of Koza's performance measure

*We find ourselves, then, met with the same difference that eternally exists between the fool and the man of sense. The latter is constantly catching himself within an inch of being a fool; hence he makes an effort to escape from the imminent folly, and in that effort lies his intelligence.*
*The Revolt of the Masses. José Ortega y Gasset*

In this chapter, the amount of uncertainty associated with Koza's performance measures is investigated. The approach used to explore this topic tries to be systematic. We identify two sources of variability (the ceiling operator and the estimation of the success probability) and study their effects. In order to simplify the analysis we decompose it into two steps. First we analyze the error in the estimation of the number of individuals to be processed, and secondly thee error in the computational effort. The analysis takes a double approach, theoretical and experimental. An analytical model of the error, based on some empirical observations, is proposed, and then it is validated by experimentation.

The main contributions of this chapter are: 1) an analytical boundary of the variability sources of Koza's performance measures validated with experimentation; 2) based on the previous result, a model for the maximum error associated to the Koza's performance measures; 3) a proposal of a new method to calculate Koza's performance measures based on the lognormality of the run-time to success. In addition, we show that a constant success probability generates a constant number of individuals to process. The conclusion that can be deduced from this work is that in common experimental settings, the error of Koza's performance measures is rather high, only a high number of runs can reduce this error to tolerable values. The use of the ceiling operator should be avoided in any case.

The chapter is structured as follows. It begins with a brief introduction and a literature review. Then, a detailed description of Koza's performance measures is presented, with an introductory discussion about the mathematical properties of the computational effort. After

that, section 6.3 introduces an exploratory experiment that also serves to motivate this chapter. Section 6.4 investigates and determines the origin of randomness in the measurement of the computational effort, including a study of the effects of the ceiling operator, and a brief overview of the effects of an error in the estimation of the success probability in the estimation of Koza's performance measures. Sections 6.5 and 6.6 are dedicated to characterize the effects of the estimation of the success probability as functions of known parameters. Then, the analytical results so far obtained are validated experimentally. We finish with some conclusions.

## 6.1 Introduction

This chapter deals with two popular measures in Genetic Programming (GP): the number of individuals to be processed to achieve at least one success with a certain probability, and the computational effort. Both, were defined by John R. Koza [136] and are closely related to each other. Computational effort is indeed obtained as the minimum value of the number of individuals to be processed. These measures do not rely on any specific element of GP, and could be used in any other generational EA, even they could be used in steady-stade algorithms with minor modifications [180]. However, perhaps for historical reasons, they only have had a relevant position in GP.

Despite the importance of the statistics proposed by Koza, and the number of research work that has been done trusting in them, the accuracy and reliability of these measures have not been object of intense investigation. Angeline first observed that the computational effort [6] is actually a random variable, and concluded that the stochastic nature of the computational effort should be handled with statistical tools. Some time after, Keijzer [127] calculated the computational effort using confidence intervals (CIs) instead of just punctual estimation, achieving a remarkable conclusion: when success probability is low, CIs of the computational effort are almost as large as the computational effort. In order words, the variability of computational effort is similar to its magnitude, and thus, in that case, the high dispersion of the computational effort makes it not reliable.

To the author's knowledge, the only systematic attempt made to understand why the computational effort presents the variability observed by Keijzer was done by Christensen [54]. He identified three sources of variability and provided empirical data that gave some light to the circumstances that reduce the reliability of computational effort. More research in this area was done by Walker [240, 241], who studied how to apply CIs to the calculus of the computational effort, and Niehaus [180], who investigated the statistical properties of the computational effort in steady-stade algorithms.

Our work in this chapter is aligned with the research done by Christensen. It is a systematic attempt to take a step forward to identify and characterize the variability sources of the computational effort. We use a theoretical and experimental approach, providing analytical boundaries to the measurement errors of Koza's performance measures, as well as experimental validation of these limits. Since the performance metrics studied in this chapter are based on the estimation of a success probability, this chapter relies on the contributions of the chapters 4 and 5.

## 6.2 Koza's performance measures

In order to clarify the terminology used in this chapter, we first define some terms that we will use. A *run* is a single execution of an evolutionary algorithm (EA), while an *experiment* is a collection of $n$ independent runs. Due to the random nature of EAs, many of their properties are stochastical, and thus they cannot be characterized using a single run, but instead an experiment with several runs. One of these properties is the *success rate* (SR), that we define as the probability of getting a success when the EA is run an infinite number of generations [21, 15]. The exact meaning of success depends on the problem and the objectives of the practitioner, so, depending on the context, we consider that a run has yielded a success if it satisfies a certain success predicate set by the experimenter. For instance, a success predicate might be finding an individual with a certain fitness value. From the point of view of SR, the exact form of this predicate is irrelevant as long as it clearly classifies the outcome of the run as "success" or "failure".

Instant and cumulative success probabilities are closely related to SR, but in contrast with it, they depend on time. Given an experiment with $y(M,i)$ successful runs in generation $i$, and each run composed by a population of $M$ individuals, the *instantaneous probability of success*, $Y(M,i)$ is defined by Koza as $Y(M,i) = y(M,i)/n$ [136]. Similarly, the *cumulative success probability* $P(M,i)$ is the cumulative success probability that derives from $Y(M,i)$, and thus we can express it as a function of $y(M,i)$ as

$$P(M,i) = \frac{1}{n}\sum_{j=1}^{i} y(M,j)$$

Expressing $P(M,i)$ as a function of the *number of successes*, $k(M,i) = \sum_{j=1}^{i} y(M,j)$ is usually more convenient, yielding that $P(M,i) = k(M,i)/n$. We should point out that $P(M,i)$ is an empirical accumulated probability, but for language abuse, it is usually ommitted.

We previously definitined SR as the accumulated success probability at the end of the experiment, and thus $SR = \lim_{i\to\infty} P(M,i)$. However, in the general case SR cannot be known, but it can be estimated. The EA is run for a fixed number of $G$ generations, $Y(M,i) = 0,\ i > G$, and $P(M,i)$ remains constant, so the estimation of SR is

$$\widehat{SR} = P(M,G) = \frac{k(M,G)}{n}$$

The definition we have made of SR implicitly assumes that there is no guarantee that the algorithm will explore all the search space, and therefore it might find a solution. It seems reasonable that, under certain conditions, for instance an EA with some types of mutation, given infinite time the algorithm will be able to find a solution [193]. This topic is open to theoretical discussion, and we simply assume that the algorithm is not guaranteed to find a solution in infinite time.

Another definition that will be useful is *generation-to-success*, which we define as the generation in which a run achieves the success [16]. Of course, this definition only makes sense for those runs that have been able to find a solution, otherwise it is not defined.

A few words should be dedicated to the notation. We use the original notation used by Koza, who expressed the cumulative success probability as a function of $M$ and $i$, where $M$

is the population size and $i$ the generation. The original intention of Koza was to emphasize the dependence of the probability with the population size and the generation. From a strict mathematical point of view, the only independent variable in the previous equation is $i$ and, with exceptions, the population size usually does not variate in the execution of an EA. The notation $P(M, i)$ might induce the idea that $M$ is an independent variable while it is a constant, hence, in our opinion, the accumulated success probability should be expressed as $P(i)$ instead of $P(M, i)$. Nevertheless, in order to be consistent, in the following we will use Koza's notation.

Another issue about notation is related to the discrete nature of EAs. The notation suggests that the performance measures are defined in continuous time, although they are discrete values. In this chapter we consider them as continuous. Our results will not be affected by this decision meanwhile the notation will be more consistent and clear. With these preliminar considerations, we are in position to introduce the computational effort.

### 6.2.1 Discussion about computational effort mathematical properties

Koza, in his classical book [136], defined a measure of the complexity of a problem and the performance of an algorithm named *computational effort*. The computational effort is based on the estimation of the number of individuals that the algorithm has to process to achieve at least one success with a given probability $z$, expressed as $I(M, i, z)$. It is common to express the probability $z$ with the greek letter $\varepsilon$ such as $z = 1 - \varepsilon$. A common value of $\varepsilon$ used in the literature is $0.01$ ($z = 0.99$).

Then, the *number of individuals to be processed* is given by

$$I(M, i, z) = M\ i\ R,$$

where $M$ is the population size and $i = 1, 2, ..., G$ the generation, thus $Mi$ is the number of individuals processed until generation $i$. Therefore $Mi$ estimates the computational processing needed to execute one run for $i$ generations, while $R$ contains the number of runs that the experiment needs to achieve, at least, one success with probability $z$, and it is

$$R = \left\lceil \frac{ln(1 - z)}{ln(1 - P(M, i))} \right\rceil \tag{6.1}$$

The operator $\lceil \ldots \rceil$ is the ceiling operator and returns the smallest integer not less than its argument, i.e., it rounds up the fractional part of its argument. This operation was introduced because $R$ gives the number of times that the experiment should be run, and thus it must be an integer. However, it should be noticed that usually it only has a mathematical interpretation, and the experiment is not supposed to be repeated $R$ times. The importance of this observation will be evident later.

Equation (6.1) can be deducted directly from Statistics and probability theory. A Bernoulli trial [174] is defined as an experiment whose outcome can take two random values, named "success" and "failure". Some problems in EC, where an optimum or near-optimum solution can be identified, may be described as a Bernoulli trial because the algorithm in those domains can achieve a satisfactory solution, or not, i.e., a "success" or a "failure" with a certain probability, the SR.

By definition, the probability of getting one success after $R$ Bernoulli trials with success probability $p$ is described by the geometric distribution,

$$P(X = R) = (1 - p)^{R-1}p,$$

and the probability of getting at least one success in $R$ trials is described by the well known cumulative distribution function (CDF) of the geometric distribution,

$$P(X \leq R) = 1 - (1 - p)^R \tag{6.2}$$

It is interesting to point out that the geometric distribution is the only discrete distribution without memory [174]. Using the CDF of the geometric distribution in Eq. (6.2) it is straightforward to calculate the number of trials $R$ such as $P(X \leq R) = z$. With these considerations we can express (6.2) with the notation used by Koza.

$$z = 1 - (1 - p)^R$$

which is the same expression that Koza deduced in [136] using probabilities. Taking natural logarithms on both sides of the equation we can isolate $R$

$$R \ln(1 - p) = \ln(1 - z) \Longrightarrow R = \frac{\ln(1 - z)}{\ln(1 - p)}$$

which is the same than (6.1), without the ceiling function. In any case, the minimum number of individuals that have to be processed to achieve at least one solution with probability $z$, takes the form

$$I(M, i, z) = Mi \left\lceil \frac{ln(1 - z)}{ln(1 - P(M, i))} \right\rceil \tag{6.3}$$

Therefore, the number of processed individuals is a function of $i$. By convenience, we define $I_c(M, i, z)$ as $I(M, i, z)$ without the ceiling operator,

$$I_c(M, i, z) = Mi \frac{ln(1 - z)}{ln(1 - P(M, i))} \tag{6.4}$$

By definition, *computational effort*, denoted by $E$, is the minimum value of (6.3),

$$E = \min_i \left\{ Mi \left\lceil \frac{ln(1 - z)}{ln(1 - P(M, i))} \right\rceil \right\} \tag{6.5}$$

Equations (6.3) and (6.5) are rather simple and easy to understand, however, understanding its behaviour and accuracy is far from being a trivial task. Several statistical issues arise when they are studied in detail, as it will be shown later. It is interesting to glimpse some of their mathematical properties before we begin to study their accuracy, variability and error sources.

### 6.2.2 Constant number of individuals to be processed

Randomness is intrinsic for all EAs, and it is also a major concern that difficults experimental as well as theoretical studies. In order to simplify it, some authors, such as Christensen [54] and Niehaus [180], used two different synthetic cumulative success probabilities that was used in their study about the properties of Koza's performance measures. The former calculated the success probability that generates a constant $I(M,i,z)$, while the latter sampled $I(M,i,z)$ from a Gaussian distribution. This last approach might not be close to the reality because of the differences that we found between that model and the data obtained empirically, as will be shown later in section 6.6.1.

The following theorem describes the relationship between $P(M,i)$ and $I(M,i,z)$, specifically, any $P(M,i)$ that corresponds to the CDF of an exponential distribution (or its discrete counterpart, the geometric distribution) generates a constant $I(M,i,z)$.

**Theorem 3** *Any cumulative probability described by the CDF of an exponential distribution, $P(M,i) = 1 - e^{-\lambda i}$ with $i \geq 0$ and $\lambda > 0$, generates a constant $I(M,i,z)$ such as*

$$I(M,i,z) = \frac{M}{\lambda} \ln \frac{1}{(1-z)} \tag{6.6}$$

**Proof 5** *The derivate of a constant function equals $0$ for all the values of the independent variable. So, and being consistent with Koza's notation, we take the partial derivate of (6.4) with respect to $i$ and equal it to $0$*

$$\frac{\partial I(M,i,z)}{\partial i} = 0,$$

*yielding the following differential equation*

$$\ln(i - P(M,i)) + i\frac{P'(M,i)}{1-P(M,i)} = 0$$

*Solving it, we obtain the function $P(M,i) = 1 - e^{\lambda i}$, where $\lambda$ is a constant parameter. We know that $P(M,i) \in [0,1]$, so we can deduce an additional condition to $\lambda$*

$$P(M,i) > 0 \Rightarrow 1 - e^{\lambda i} > 0 \Rightarrow \lambda < 0$$

*Given that $i$ is a natural number, any success probability $P(M,i)$ that generates a constant computational effort must be of the form*

$$P(M,i) = 1 - e^{-\lambda i},\ \lambda \in \mathcal{R}^+ \tag{6.7}$$

*which corresponds to the CDF of the exponential distribution.*

The equation that Christensen and Oppacher deduced in [54] can be obtained from (6.6). It is interesting to note that the exponential distribution is the only memoryless distribution, it means that, in some sense, the variability of $I(M,i,z)$ is a consecuence of the memory of the accumulated success probability.

Table 6.1: Best estimation of success probability for the artificial ant problem. It reports the number of runs ($n$), number of successful runs ($k$), best estimation of success rate $\hat{P}^{best}(M,G)$, best estimation of computational effort ($\hat{E}^{best}$), best estimation of computational effort without ceiling operator ($\hat{E}_c^{best}$) and their difference in absolute as well as relative values.

| | Artificial ant | 6-Multiplexer | 5-Parity | Regression |
|---|---|---|---|---|
| $n$ | 100,000 | 100,000 | 5,000 | 100,000 |
| $k$ | 13,168 | 95,629 | 305 | 29,462 |
| $\hat{P}^{best}(M,G)$ | 0.13168 | 0.95629 | 0.061 | 0.29462 |
| $\hat{E}^{best}$ | 490,000 | 24,000 | 14,800,000 | 117,000 |
| $\hat{E}_c^{best}$ | 487,276 | 22,805 | 14,633,571 | 116,468 |
| Difference | 2,724 (0.5%) | 1,195 (4.98%) | 166,429 (1.13%) | 536 (0.49%) |

In any case, the deterministic function studied here is interesting from a theoretic point of view, but actually in real EA experiments the behaviour of $P(M,i)$ and $I(M,i,z)$ is much more complex. An initial insight to this behaviour is given in the next section, which introduces an exploratory experiment showing an initial statistical overview of computational effort.

## 6.3 Exploratory experimental analysis

There are two main problems concerning the experimentation that we have to carry out in this chapter. First, since we are interested in the accuracy of the measures under study, there is a need to have something to compare with, to take as reference; ideally it should be the exact measure, but clearly it is not possible. Secondly, we need a high number of algorithm runs, with a high consumption of computing resources. These two problems can be solved using resampling methods.

Like in chapter 4, four classical GP study cases have been selected: Artificial ant with the Santa Fe trail, 6-multiplexer, even 5-parity and a linear regression [136]. They have been selected to represent a diversity of difficulties, from an easy problem (6-multiplexer) to a difficult one (5-parity), with two intermediate problems (artificial ant and regression). Each one of these domains was run a high number of times, $100,000$, with the exception of the 5-parity, that was only run $5,000$ times because its greater population size required more computational resources. The main advantage that it provides is that using all the runs it is possible to calculate an accurate estimation of the metrics under study. A second advantage is that once those runs are executed and stored, they can be resampled to avoid running again the algorithms, saving substancial computational resources and time.

The object of this study is not the algorithm itself, but rather the performance metrics, so the details of their implementation and the parameter tuning does not affect this study. Consequently, we have used the default implementation of the selected problems and parameters

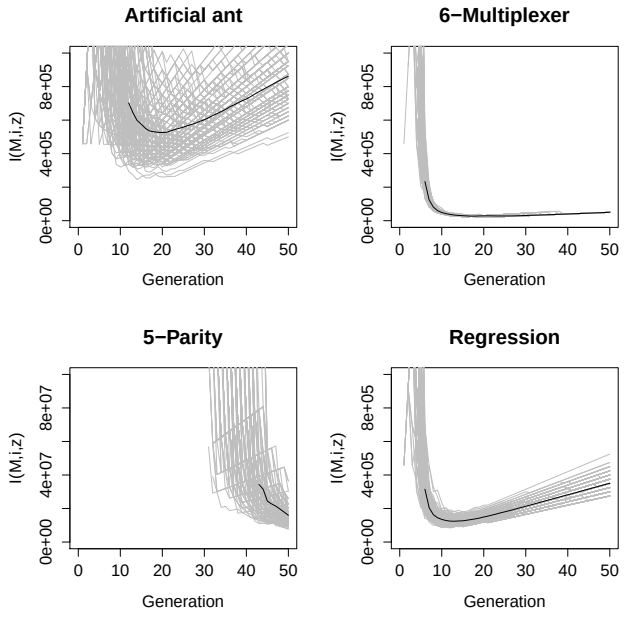


Figure 6.1: Number of individuals to be processed of $200$ pseudoexperiments composed by $50$ runs. The mean value is plotted with a black solid line.

found in ECJ v18 [154], which are based on the original settings used by Koza [136]. The main parameters that we have used are reported in Table 4.2, with just minimal corrections such as the population size. The large number of runs executed yields a good estimation of the true values of Koza's metrics. Since they are the best estimation available for this study, we note them as $\hat{I}^{best}(M,i,z)$, $\hat{P}^{best}(M,i)$ and $\hat{E}^{best}$. The values of these estimations are shown in Table 6.1.

The variability of the estimation of $I(M,i,z)$ is depicted in Figure 6.1. It contains the outcome of $200$ simulated experiments (or pseudoexperiments) with its average value. Each experiment has been simulated taking $50$ samples with replacement from the dataset. This figure shows that different pseudoexperiments usually yield different performance curves. Depending on the domain, the variability of the curves changes, for instance, if we compare the curves of the artificial ant and the 6-multiplexer, we find less variability in the latter than in the former. Notice that the scale used in the figure in both cases is the same.

At this point it makes sense for us to hypothesize that the problem difficulty plays a role, this hypothesis is based on the apparent correlation between the success rate of each problem and the dispersion of their $\hat{I}(M,i,z)$ curves. The two most difficult problems, the artificial ant and the 5-parity, are those with greater variability whereas the two easiest problems, 6-multiplexer and the regression, present less variability.

Figure 6.2 shows the histograms of the computational effort calculated for the problem domains under study. Each histogram uses $5,000$ pseudoexperiments calculated using $50$ (bottom row), $200$ (middle row) and $500$ (top row) runs sampled from the datasets of runs. Histograms do not clearly suggest a distribution function able to fit data in all the cases. Computational effort in the regression problem takes a triangular form while, for instance,

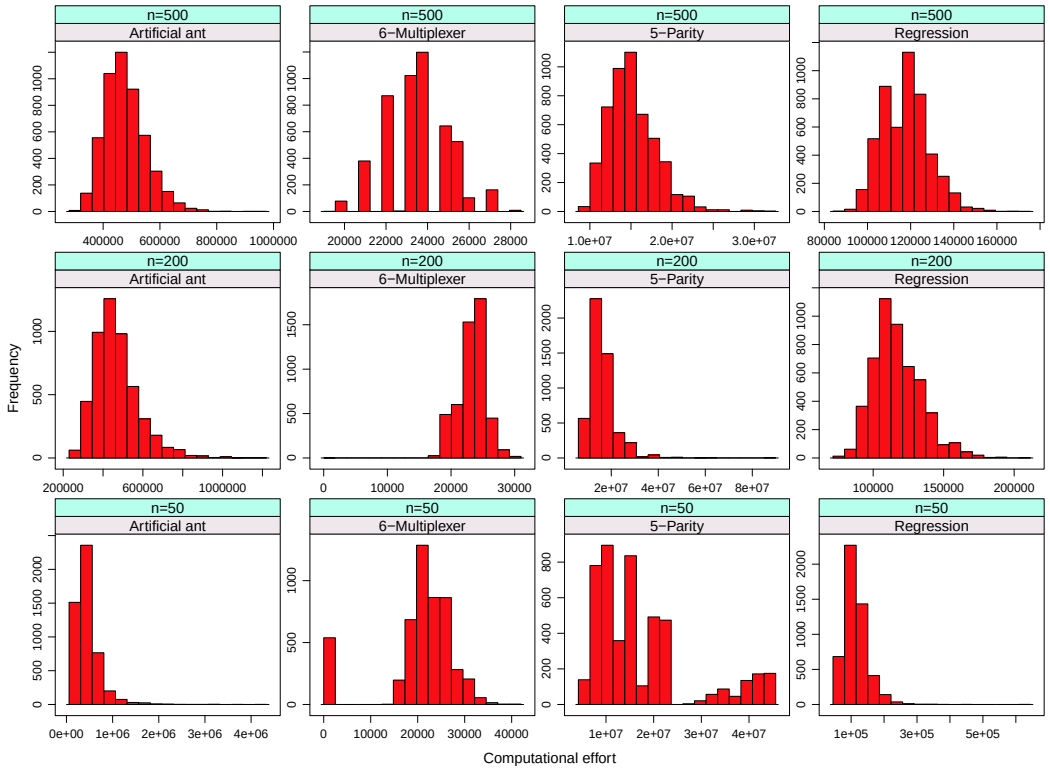


Figure 6.2: Histograms of the computational effort for the four problems under consideration. Each histogram represents the computational effort of $5,000$ experiments that were simulated subsampling $50$, $200$ and $500$ runs from the datasets.

the artificial ant seems to fit better in a lognormal or a Weibull distribution. There are also some outsider histograms, such us the parity problem for $n = 50$ or the multiplexer with $n = 50$, nonetheless, the latter can be explained by the grouping of the categories in the histogram.

The lack of an obvious distribution able to describe $\hat{E}$ confirms the previous result reported by Walker *et al* in [241], who also failed in finding a probability distribution able to model $\hat{E}$. In our opinion, there is an underlying random variable associated to the accumulated success probability, and this random variable is modified by several non-lineal operations such us logarithms and the minimum operator, so $\hat{E}$ in some sense follows the same distribution but it has been "contaminated" by those operations. From another point of view, differences in the distribution of $\hat{E}$ with different levels of $n$ suggest the presence of a sampling bias [59], and thus the presence of other factors that influence $E$. We suspect these factor are the non-linear operations made by (6.3) and (6.5).

An important property of any estimator is its variability. Figure 6.2 illustrates a relationship between the variability of the estimator and the number of runs: the higher is $n$, the narrower is the distribution of $\hat{E}$. Let us, for instance, observe the artificial ant, when $n = 50$ most of the estimators are placed between $0$ and $1.5E6$ individuals, if we increase the number of runs to 200, most of $\hat{E}$ take values between $200,000$ and $800,000$; higher val-

ues of $n$ yield even less variability of the estimations of computational effort, when $n = 500$ $\hat{E}$ is mostly placed in the range of $300,000$ and $700,000$. This behaviour is observed also in the rest of problem domains. Since the estimation of $I(M, i, z)$ and $E$ depends on the estimation of the accumulated probability, and its quality is highly dependent on the number of runs [15], it makes sense to suppose that they are related to each other.

In any case, it is clear that performance measures contain a variability that comes from the intrinsic stochastic nature of experimentation. However the exact nature of the variability and the factors that influence its magnitude is not yet clear, so we move on to try to identify the sources of that variability in order to quantify and model it.

## 6.4 Determination of the variability sources

Previous work performed by Christensen identified three sources of variability in computational effort, the ceiling operator, the estimation of the success probability and the minimum operator [54]. We hold a slightly different point of view about the effects of the minimum operator. First of all, we hold that, to be strict, it is necessary to clearly distinguish between $I(M, i, z)$ and $E$, something that some studies do not do. In our opinion, the minimum operator is a deterministic non-linear operator whose reliability depends on its operand. In other words, the reliability of the measurement of $E$ only depends on the measurement of $I(M, i, z)$, which does not depend on any minimum operator. So we explicitly exclude the minimum operator in the study, and we will simply study the reliability of $I(M, i, z)$, and then apply the result to the computational effort.

Consequently, we consider two sources of randomness in $I(M, i, z)$ and $E$: the ceiling operator and the estimation of the cumulative success probability. In order to simplify the study of the effects of these uncertainty sources, we separate them as independent noise sources using the following model.

$$I(M, i, z) = \hat{I}_c(M, i, z) + \varepsilon^I_{ceil} + \varepsilon^I_{est}$$

while, by definition,

$$E = \min_i \left(I(M, i, z)\right) = \min_i \left(\hat{I}_c(M, i, z) + \varepsilon^I_{ceil} + \varepsilon^I_{est}\right)$$

So we can identify an uncertainty $\varepsilon^I_{ceil}$ generated by the ceiling operator, as well as a noise $\varepsilon^I_{est}$ associated to the estimation of $P(M, i)$, which is related to the limitation in the number of runs. This chapter moves towards characterize $\varepsilon^I_{ceil}$ and $\varepsilon^I_{est}$, to try to understand how they affect the precision of performance measures, and then look for methods to reduce its variability while improving the reliability.

### 6.4.1 Ceiling operator

The first variability source we study is the ceiling operator. Strictly speaking, the ceiling operator is not a randomness source because it is a deterministic operator, but it removes information, increases the variability of the measure and reduces its precision, as will be demonstrated later, so its effects in practical terms is the same than a biased random error.

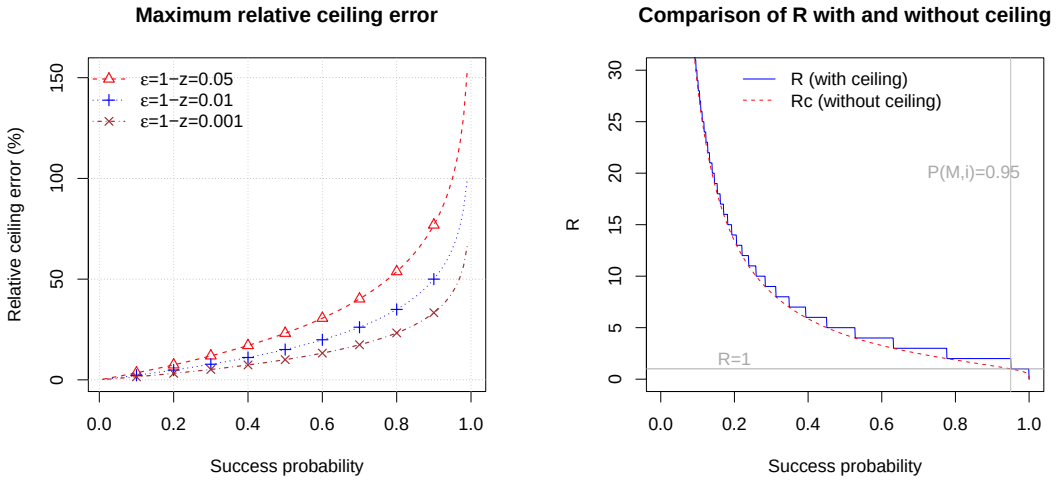


Figure 6.3: Maximum relative ceiling error as function of success probability for three values of $\varepsilon$, $0.05$, $0.01$ and $0.001$ (left) and comparison between $R$ and $R_c$ (right).

In order to study the effects of the ceiling operator, let us define an alternative computational effort, $E_c$, as $E_c = \min_i (I_c(M,i,z))$, where $I_c(M,i,z) = MiR_c$ and

$$R_c = \frac{ln(1-z)}{ln(1-P(M,i))}$$

It is clear that, assuming that $\varepsilon_{est} = 0$, the error term introduced by the ceiling operator is the difference between $I(M,i,z)$ and $I_c(M,i,z)$,

$$\varepsilon^I_{ceil} = I(M,i,z) - I_c(M,i,z) = Mi(R - R_c)$$

The error depends on the the fractional part of $R$ that is rounded by the ceiling operator, the population size and the generation number. We know that $(R - R_c)$ is limited by the maximum fractional part of a real number, so $(R - R_c) < 1$. With this consideration, it is possible to bound $\varepsilon^I_{ceil}$

$$\varepsilon^I_{ceil} < Mi \tag{6.8}$$

It follows that $\max(\varepsilon^I_{ceil}) = Mi$. This equation introduces an absolute limit to the ceiling error which is linear with $i$ for a given population size. One way to study the importance of this error compared with $I(M,i,z)$ is calculating the relative ceiling error ($\varepsilon^I_{ceil}(\%)$), which is straightforward using the definition of $I_c(M,i,z)$ and (6.8).

$$\varepsilon^I_{ceil}(\%) \leq \frac{\max(\varepsilon^I_{ceil})}{I_c(M,i,z)} = \frac{\ln(1-P(M,i))}{\ln(1-z)} \tag{6.9}$$

and thus the relative maximum ceiling error in the measure is function of $P(M,i)$ and $z$. A graphical representation of (6.9) for three common $z$ values can be found in Figure 6.3

(left). It can be seen that the estimation of $I(M,i,z)$ is more sensitive to the ceiling error for low values of $z$. Figure 6.3 (left) also clearly shows the asymptotic behaviour of the maximum relative ceiling error next to 1, so when $P(M,i)$ is close to 1, the estimation of $I(M,i,z)$ might become arbitrary wrong. A graphical comparison of $R$ and $R_c$ can be found in Figure 6.3 (right).

Given the reported results in this section, we conclude that the error generated by the ceiling operator might be very significant. The maximum amount of error introduced by this operator depends on the population size and the generation, nevertheless, in relative terms, it only depends on the accumulated success probability and $z$. We do not recommend using values of $z$ lower that $P(M,i)$. The good news are that, once $P(M,i)$ is known, the error may be bounded by (6.9), moreover, it can be completely eliminated simply removing the ceiling operator. We are unable to find any remarkable disadvantage, so, evidence moves us to suggest not using the ceiling operator when calculating $I(M,i,z)$ and $E$. In few words, ceiling error might be a considerable source of variability in the measures, however it is trivial to remove it in comparison with the estimation error, which is studied in the next section.

### 6.4.2 Estimation error

The second source of variability we can identify comes from the estimation of $P(M,i)$. The true success probability is rarely known in EC, and therefore the experimenter has to estimate it [54]. In practical terms, (6.3) cannot be directly used, but rather we can obtain an estimation

$$\widehat{I}(M,i,z) = Mi \left\lceil \frac{ln(1-z)}{ln(1-\widehat{P}(M,i))} \right\rceil \tag{6.10}$$

where $\widehat{P}(M,i)$ is the estimation of the accumulated success probability obtained from the experiments

$$\widehat{P}(M,i) = \frac{k(M,i)}{n}$$

The difference between the theoretical $P(M,i)$ and the experimental $\hat{P}(M,i)$ is the only randomness source of $I(M,i,z)$, the error induced by this difference is what we call *estimation error*.

Following [54], we model the estimation error $\varepsilon_{est}$ as a noise $P(M,i) = \hat{P}(M,i) + \varepsilon_{est}$. This error, associated to the estimation of $P(M,i)$, induces another error in the estimation of $I(M,i,z)$, that we model adding an error term $\varepsilon^I_{est}$, so $I_c(M,i,z) = \hat{I}_c(M,i,z) + \varepsilon^I_{est}$. In order to isolate the effects of the estimation error and avoid unnecessary complexity, we do not consider the ceiling operator.

With these considerations we can state that the number of individuals to be processed is given by

$$I_c(M,i,z) = Mi\frac{\ln(1-z)}{\ln(1-(\hat{P}(M,i)+\varepsilon_{est}))} = Mi\frac{\ln(1-z)}{\ln(1-\hat{P}(M,i))} + \varepsilon^I_{est},, \tag{6.11}$$

then, the estimation error of $I_c(M,i,z)$ as a function of the estimation error of the cumulative

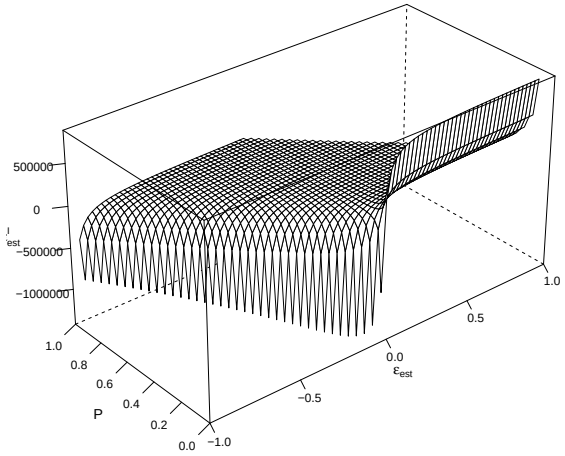


Figure 6.4: Error produced by the estimation of $P(M,i)$ as function of success probability and estimation error. The function is defined by $P \in (0,1]$, $\varepsilon_{est} \in [-1,1] \setminus P + \varepsilon \in (0,1]$. $M$ is fixed to 500, $i$ to 10, and $\varepsilon = 0.01$.

success probability is the difference between $I_c(M,i,z)$ and $\hat{I}_c(M,i,z)$

$$\varepsilon^I_{est} = I_c(M,i) - \hat{I}_c(M,i) = \frac{Mi\ln(1-z)}{\ln(1-P(M,i))} - \frac{Mi\ln(1-z)}{\ln(1-(P(M,i)+\varepsilon_{est}))} \tag{6.12}$$

To ease the graphical representation of (6.12) shown in Figure 6.4 we consider $P(M,i) = P$ and $\varepsilon_{est}$ as independent variables, fixing the rest of the parameters to some common values, $M = 500$, $\varepsilon = 0.01$ and $i = 10$. Figure 6.4 shows that $\varepsilon^I_{est}$ has an asymptotic behaviour in two planes, $P = 0$ and $\varepsilon_{est} = P$.

The high estimation error found in the plane $P = 0$ was previously observed by Christensen [54]. He calculated the Taylor series of (6.11) and found that $I(M,i,z)$ is very sensitive to estimation errors when $P(M,i)$ is close to 0, which is the situation in early generations of the evolutionary process. The reason of this sensitive area can be found in the first term of (6.12), given $\varepsilon_{est} \neq 0$, and taking the limit

$$\lim_{P(M,i)\to 0} \left( \frac{Mi\ln(1-z)}{\ln(1-P(M,i))} - \frac{Mi\ln(1-z)}{\ln(1-(P(M,i)+\varepsilon_{est}))} \right)$$

yields an infinite error. The another asymptotic behaviour of the estimation error is originated by the second term of (6.12). When $\varepsilon_{est} \approx -P(M,i)$ the denominator tends to be 0, and then the estimation error increases its magnitude. It should be noticed that this effect is not symmetrical, only happens for negative values of $\varepsilon_{est}$, i.e., when $P(M,i)$ is overestimated.

The relative estimation error is given by the cocient $\varepsilon^I_{est}/I_c(M,i,z)$, then, using the definition of $I_c(M,i,z)$ and (6.12)

$$\varepsilon^I_{est}(\%) = \frac{\varepsilon^I_{est}}{I_c(M,i,z)} \leq 1 - \frac{\ln(1-P(M,i))}{\ln(1-(P(M,i)+\varepsilon_{est}))} \tag{6.13}$$

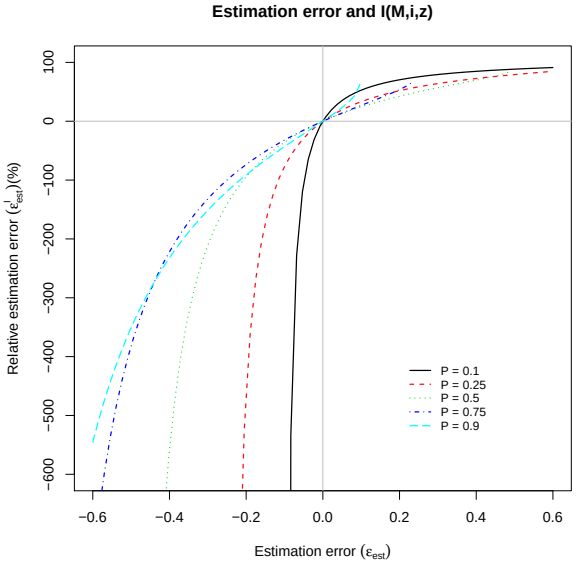


Figure 6.5: Relative error as function of SR and estimation error. The function is defined by $P \in (0,1], \varepsilon_{est} \in [-1,1] \backslash P + \varepsilon_{est} \in (0,1]$.

This equation provides a way to determine whether the error is significant in relative terms. When the estimation error is small, the ratio is close to $1$ and therefore the estimation error is, in proportion, close to $0\%$.

The relationship between $\varepsilon_{est}$ and $\varepsilon^I_{est}$ given by (6.13) can be better appreciated in Figure 6.5. First we observe that the sign of $\varepsilon_{est}$ has a strong influence on $\varepsilon^I_{est}(\%)$, an overestimation of $P(M,i)$ (negative $\varepsilon_{est}$) leads to more error in the calculus of $I(M,i,z)$ than an underestimation (positive $\varepsilon_{est}$) of the same magnitude. This behaviour is explained by the asymptotic effects of the relative estimation error when $P(M,i) + \varepsilon_{est} = 0$, which was seen before in Figure 6.4. Secondly, it can be seen that $\varepsilon^I_{est}(\%)$ also depends on the value of $P(M,i)$; low values of $P(M,i)$ are more sensitive to the estimation error than high ones, as can be seen in the slope of the curves, much more inclined in the former case.

As a conclusion, we can state that the number of processed individuals is specially sensitive to the estimation error in two cases: when the accumulated success probability is very low, close to $0$, and when $\varepsilon_{est} \approx -P(M,i)$. Drawing conclusions about the relationship between the estimation error and $E$ requires further analysis, which basically deals with the minimum operator. In any case, and as an almost tautological conclusion, high estimation errors will generate high errors in the calculus of the number of processed individuals, and we can conjecture that this error will also be translated to the estimation of the computational effort.

In this section we have related the estimation error, $\varepsilon_{est}$, with the error that it introduces in $\varepsilon^I_{est}$, however, it is yet unclear what factors determine the magnitude of $\varepsilon_{est}$.This is not a minor observation, depending on the value of $\varepsilon_{est}$ the error induced might be significant or not. We need to express that error as function of a known measure. That is the objective of the next two sections, model $\varepsilon_{est}$ as function of known factors and check out its influence on

the accuracy of $I(M, i, z)$ and $E$.

## 6.5 Characterization of the estimation error of $I(M, i, z)$

The magnitude of the estimation error of $P(M, i)$ is a key element to explain the accuracy of Koza's performance measures. Due to the stochastic nature of $\varepsilon_{est}$, it is not possible to set a hard limit to its size, as was done with the ceiling error (6.8), nonetheless, it does not mean that there are no mathematical tools that could give some light to this topic. The study on the binomiality of SR and confidence intervals done in chapter 4 is an example.

The maximum likelihood estimator of the accumulated success probability is $\hat{P}(M, i) = k(M, i)/n$. Given a certain generation, let say $i_o$, the number of successful runs $k(M, i_0)$ is a binomial random variable [21, 15, 177]. Then $\varepsilon_{est}$ is an error associated to the estimation of a binomial variable, and thus, it can take any value between $0$ to $1$. Nonetheless, it is still possible to determine a region where the estimation of the probability is likely to be contained with CIs [44, 179].

The *Confidence Interval Width* (or CIW) is defined as the difference between the upper and lower bound of the interval, so $CIW = U - L$, we can set a relationship between the CIW and the estimation error, as it will be justified later. These two properties, CP and CIW, are commonly used to measure the quality of an interval [179]. A good interval has a CP close to the nominal coverage and low CIW.

Binomial CIs is a well studied problem, and there is a large corpus of publications dealing with this topic. Probably, the two best known comparative studies about binomial CIs are [44, 179], but many other studies have been published [45, 235, 190]. Much less research have been done to relate this field of Statistics with EC, however using CIs in the context of computational effort is not a new idea. Walker *et al.* studied how to apply CIs to the estimation of the computational effort [241], the reliability reliability of CIs [240] and proposed a new metric called "success effort" using a binomial CI [242]. Niehaus and Banzhaf also studied the reliability of the computational effort [180] within steady-stade algorithms.

Probably the main problem here is the dependence of $P(M, i)$ with $i$. For a fixed generation $i = i_0$, the cumulative number of successes $k(i_0)$ can be described using a binomial distribution, and thus it is straightforward to create a CI for the probability $P(M, i_0)$, but if we introduce the generation number, $P(M, i)$ is no longer a binomial random variable, but rather a stochastic process. This topic was addressed in detail in chapter 5. Walker [240] proposed some solutions, but finally concludes that the best performance is achieved when the CI is calculated for each generation, and therefore, given the CI of the accumulated success probability in generation $i$, $[L_i, U_i]$, the CI of $I(M, i, z)$ is

$$\begin{aligned} L_i^I &= Mi \left\lceil \frac{ln(1-z)}{ln(1-L_i)} \right\rceil \\ U_i^I &= Mi \left\lceil \frac{ln(1-z)}{ln(1-U_i)} \right\rceil \end{aligned} \tag{6.14}$$

If the algorithm is run for $G$ generations, calculating intervals for $I$ using (6.14) requires $G$ CIs.

Following Walker *et al.*, computational effort is then the interval $[L_j^I, U_j]$, where $j$ is the generation where $I(M, i, z)$ achieves its minimum. As Walker noticed, this method does not consider that $j$ is a random variable, and thus we do not feel this is the most accurate method to calculate computational effort. We need to know how $P(M, i)$ varies with the generation.

Any binomial CI method may be used, for instance, a normal approximation or Wald interval [145], Clopper-Pearson or "exact" interval [58], Agresti-Coull or adjusted Wald [2], Wilson or 'score' [247], not to mention alternative bayesian approaches [91, 214]. No matter which method is used, binomial CIs can be used to characterize the magnitude of the estimation error.

### 6.5.1 Relative error induced by the estimation error in $I(M.i, z)$

There are several binomial CI methods, each one with its own properties. However, we have seen in chapter 4 that their basic properties are common for all the methods, and only a through analysis of the methods may find differences in their behaviour. In any case, for the purpose of this research, these differences are not significant, we are interested in the common properties of binomial CIs to characterize $\varepsilon_{est}$, not in the particularities of each method. Several authors [21, 240, 44] recommend the classical Wilson method. This method combines good performance in average term with simplicity, these properties makes Wilson a good choice to characterize $\varepsilon_{est}$, these observations were confirmed in chapter 4. For this reason, we select the method of Wilson with continuity correction, that corrects some aberrations found in the original method [45], so the form of the CI used in this study is

$$\begin{aligned} L &= \frac{k + \frac{1}{2}z_{\alpha/2}^2}{n + z_{\alpha/2}^2} - \frac{z_{\alpha/2}^2\sqrt{n}}{n + z_{\alpha/2}^2}\sqrt{p(1-p) + \frac{z_{\alpha/2}^2}{4n}} \\ U &= \frac{k + \frac{1}{2}z_{\alpha/2}^2}{n + z_{\alpha/2}^2} + \frac{z_{\alpha/2}^2\sqrt{n}}{n + z_{\alpha/2}^2}\sqrt{p(1-p) + \frac{z_{\alpha/2}^2}{4n}} \end{aligned} \tag{6.15}$$

where $p = k/n$ is the maximum likelihood estimator of the success probability, $k$ is the number of successes, $n$ is the number of runs and $z_{\alpha/2}$ is the upper-$\alpha/2$ critical point from $N(0, 1)$.

It should be noticed that the center of the interval is not given by $p = k/n$, but rather by $\tilde{p} = (k + \frac{1}{2}z_{\alpha/2}^2)(n + z_{\alpha/2}^2)^{-1}$, hence the punctual estimator $\hat{P}(M, i)$ is not placed in the center of the interval. This is a common characteristic of almost all the binomial CIs [44] produced by the boundaries of the random variable. From the point of view of the calculus of $I(M, i, z)$ the shift of the location of $p$ within the interval produces that a low success rate close to $0$ is more likely to have underestimated its value, because there are no negative probabilities. On the contrary, an estimation close to $1$ probably will overestimate the real SR because there are no probabilities greater than $1$. This fact has to be considered to obtain a fair estimation of the error, actually, a quantification of this asymmetry is needed.

The effects of the asymmetry of the estimation of a probability can be studied using the distance between $\hat{p}$ and the boundaries of the interval $[L, U]$. Inspired by Newcombe [179], and as a way to measure the asymmetry of the interval, we define the *Distal Confidence Interval Width*, or DCIW, as the difference between the maximum likelihood estimator of

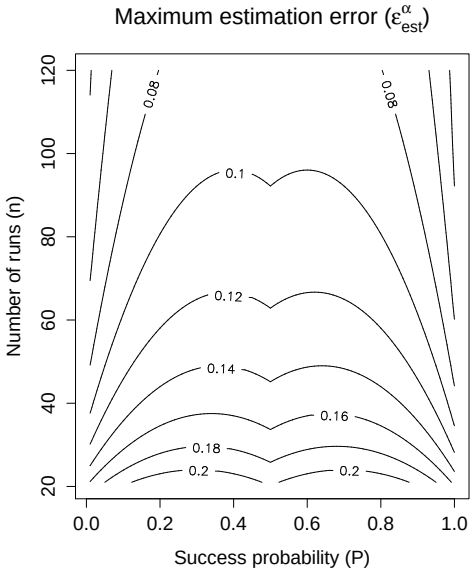


Figure 6.6: Maximum estimation error $\varepsilon^{\alpha}_{est}$ as function of number of runs and success probability. $\varepsilon^{\alpha}_{est}$ has been calculated as the maximum of Wilson DCIW and MCIW for $\alpha = 0.05$ (see Eq. (6.16)).

the probability and the lower limit, so, $DCIW = \hat{p} - L$. Similarly, we define the *Mesial Confidence Interval Width*, or MCIW, as the difference $MCIW = U - \hat{p}$. It is trivial to demonstrate that CIW, DCIW and MCIW satisfy the property $CIW = DCIW + MCIW$. Differences between DCIW and MCIW tend to be reduced when there is a high number of runs.

If the probability estimator was centered in the interval, we could set a direct relationship between the estimation error and the CIW, simply $\varepsilon^{\alpha}_{est} \leq CIW/2$. This boundary for the estimation error is expected to be true with probability $1-\alpha$. Nonetheless, the asymmetry of the binomial distribution does not ensure a fair estimation of the error just taking the center of the interval. Good CIs methods should guarantee more or less the same probability of underestimate and overestimate $p$ within the interval [179]. However their effects are not the same, because DCIW and MCIW are not likely to be equal (it only happens when $p = 0.5$), and thus the error that is associated, following that $\varepsilon^{\alpha}_{est} \leq CIW/2$ actually underestimates the error. In order to be conservative, and to obtain a more fair estimation of the error, we take the maximum between DCIW and MCIW, so

$$\varepsilon^{\alpha}_{est} \leq \max(DCIW, MCIW) \tag{6.16}$$

To better illustrate the properties of (6.16), it has been depicted in Figure 6.6. The error $\varepsilon^{\alpha}_{est}$ is symmetrical wrt the axis $P = 0.5$, however its maximum is not placed on that axis, but it is biased, the reason can be found in the displacement of $\hat{p}$ with respect to the center of the interval. Interestingly, the effect of the asymmetry of the interval is less evident as the number of runs is increased and rather obvious for low values of $n$. In any case, higher number of samples always generates tighter intervals because there is more information about the

algorithm [21], and thus more precise intervals can be built and therefore the estimation error is reduced.

Looking only CIW may lead to an incomplete view of the CI performance. In particular, evidence shown previously may induce the (incorrect) conclusion that estimation of probabilities close to $0$ and $1$ are more reliable than those with intermediate values because in that area intervals are tighter. The confidence level $\alpha$ is also known as nominal coverage because intervals cover the true probability with probability $(1-\alpha)$, nevertheless, the real coverage that one finds in practice is unlikely to coincide with the nominal one. This is the reason because it is necessary to look at CP. Figure 4.4 shows the CP for Wilson intervals with $\alpha = 0.05$. Like CIW, CP is symmetrical with $P = 0.5$ and explains why intervals are tighter near the boundaries of the domain, CP in those areas gets worse and actually, for values of $p$ close to $0$ and $1$ the intervals are not reliable at all. Intervals are tighter, but it is more unlikely that the estimated probability falls within the interval. In these regions, when number of runs is large, successes are better described using a Poisson distribution [148] instead of a binomial.

In this section we have developed some tools in order to help us to understand under which circumstances the estimation error is higher. However, the question about how these circumstances affect the estimation of $I(M,i,z)$ and $E$ remains open. Next section tries to provide some light to the answer.

### 6.5.2 Relative magnitude of $\varepsilon^I_{est}$

The main variability source of $I(M,i,z)$ is the estimation of $P(M,i)$. Due to the intrinsic stochastic nature of the estimation, it is necessary to use statistical methods to characterize its behaviour. Confidence intervals provides a tool to estimate $\varepsilon_{est}$ as a function of the number of runs and the success probability. Since $\varepsilon^I_{est}$ is a function of $\varepsilon_{est}$, we can use the previous result to directly develop a relationship among $\varepsilon^I_{est}$, $n$ and $p$.

With all these considerations, we can limit the effects of the estimation error in the measurement of $I(M,i,z)$ as

$$\varepsilon^I_{est} \leq \frac{1}{2}\left( Mi\frac{\ln(1-z)}{\ln(1-L_i)} - Mi\frac{\ln(1-z)}{\ln(1-U_i)} \right) \tag{6.17}$$

where $[L_i, U_i]$ follows (6.15).

In order to ease the analysis of the relative effects of the estimation error, we calculate the relative maximum error $\varepsilon^I_{est}(\%)$ as the ratio of the maximum error and the number of individuals to be evaluated with the probability placed in the center of the interval, $\tilde{p} = (k + \frac{1}{2}z^2_{\alpha/2})(n + z^2_{\alpha/2})^{-1}$.

$$\varepsilon^I_{est}(\%) = \frac{\varepsilon^I_{est}}{I(M,i,z)} \leq \frac{\ln(1-\tilde{p})}{2}\left( \frac{1}{\ln(1-L_i)} - \frac{1}{\ln(1-U_i)} \right)$$

The surface defined by this equation for $\alpha = 0.05$ is depicted in Figure 6.7. It shows that $\varepsilon^I_{est}(\%)$ is highly dependent on the number of runs and success probability. Using a high number of runs yield less error in the measurement of $I(M,i,z)$. The influence of the success probability is slightly more complicated. Low values of $P(M,i)$ yields poor

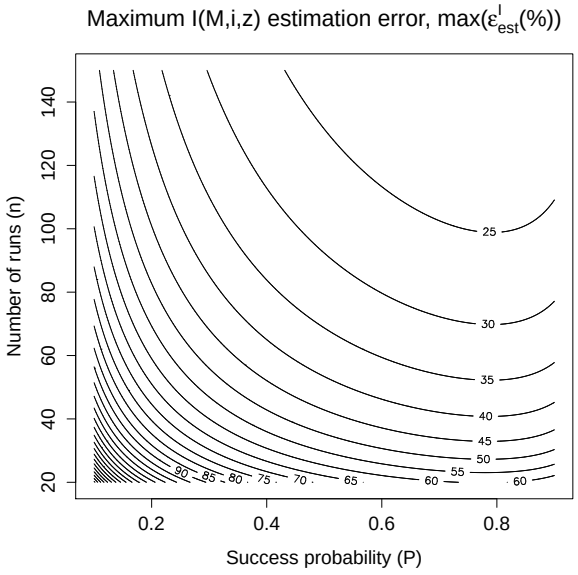


Figure 6.7: Maximum relative error of $I(M, i, z)$ calculated using confidence intervals with $\alpha = 0.05$. X-axis represents the success probability ($p$) whereas y-axis represents the number of runs ($n$).

estimations of $I(M, i, z)$, the same can be said when $P(M, i)$ is close to 1, however, in this case the effect is not so evident. It is interesting to note, again, that early generations of the EA, where the success probability is lower, the estimation error is greater, and thus in that region the estimation of $I(M, i, z)$ looses accuracy. The evolution of an EA may be represented graphically in Figure 6.7 as a point that moves from left to right, the first generation is placed in $(0, n)$ and moves horizontally to $(\widehat{SR}, n)$ in $G$ generations.

Let us consider a common experimental setup composed by $60$ runs, a value within an order of magnitude commonly used in practice. Looking at Figure 6.7 we find that the maximum relative estimation error is, at least, around $34\%$ when $P(M, i) = 0.78$. This error does not include the error produced by the ceiling operator, and actually, it is not guaranteed that the experiment achieves it, for instance, in case that the SR achieved by the algorithm were lower than $0.78$. It shows that the estimation error is rather significant even for a relative high number of runs, $60$, moreover, finding literature reporting fewer number of runs is not rare.

How the estimation error affects the computational effort is far from being a trivial problem. Computational effort is the minimum of $I(M, i, z)$, which is a deterministic non-linear operation and thus it does not introduce randomness in the measure. An issue that makes difficult the analysis of $E$ is its dependence with the generation number $i$, this dependence affects in two ways. Indirectly, through $P(M, i)$, which is a function of $i$, and directly because it is included in (6.3). Another factor that difficults the study of $E$ is that $P(M, i)$ cannot longer be considered as a point estimator, but rather as a stochastic process because it depends on $i$, i.e., there is a statistical dependence between $P(M, i)$ and $i$. This issue is disscussed in detail in the next section.

## 6.6 Characterization of the estimation error of $E$

The main difficulty that we find in the study of $E$ is that, on the contrary than $I(M,i,z)$, its statistical properties do not depend on an underlying binomial random variable, but on a stochastic process. Therefore, we need a model of the time-behaviour of $I(M,i,z)$, which also depends on the existence of a model of success probability. Fortunately, this problem was addressed in the previous chapter and therefore we are in position to develop a model of the computational effort.

### 6.6.1 Analytical model of $I(M,i,z)$ and $E$

Equation (5.19) provides a reasonable model of the accumulated success probability in tree-based GP, as it was demostrated in section 5.4.3). It lead us to an alternative method to calculate $I(M,i,z)$, let us name this method $I^\star(M,i,z)$. Using (5.15) and (6.3) we trivially obtain that the estimator of $I^\star(M,i,z)$ is given by

$$\hat{I}_c^\star(M,i,z) = Mi\frac{\ln(1-z)}{\ln\left(1-\frac{k(M,G)}{n}\Phi\left(\frac{\ln i-\hat{\mu}}{\hat{\sigma}}\right)\right)} \tag{6.18}$$

where $\Phi(...)$ is the standard normal CDF [125], $\hat{\mu}$ is given by (5.17) and $\hat{\sigma}$ by (5.18). Then, $\hat{I}_c^\star(M,i,z)$ is a function of three parameters, $k(M,G)$, $\hat{\mu}$ and $\hat{\sigma}$. Once that we have $I^\star(M,i,z)$, providing an analytical model of $E$, let us name it $E^\star$, is straitforward. The estimator of $E^\star$ is then

$$\hat{E}^\star = \min\left\{Mi\frac{\ln(i-z)}{\ln\left(1-\frac{k(i)}{n}\Phi\left(\frac{\ln i-\hat{\mu}}{\hat{\sigma}}\right)\right)}\right\} \tag{6.19}$$

A graphical representation of the proposed model of $E$ is given by Figure 6.8. Given that the definition of $E$ contains a minimum operator, it is not unreasonable to conjecture that $E$ depends on points close to the minimum operator that could alter the estimation. On the contrary than $\hat{E}$, $\hat{E}^\star$ uses all the sampled points, and thus it seems reasonable to hypothetize that the estimation made is less sensible to local outliers, yielding a more robust estimator. This is just an hyphothesis, and thus it is desirable to confirm it through experimentation. Similarly, we have proposed a model, but we do not know if this model is able to approximate $E$ well. These two issues are addresses in the following section.

### 6.6.2 Experimental validation of the analytical model of $E$

The model proposed in (6.19) is an alternative formulation of the computational effort, and, in order to be useful in this study, it should be able to estimate $E$ reasonably well. A side effect of the definition of $E^\star$ is that it provides an alternative method to calculate computational effort that uses all the available samples, and then it seems seasonable to hypothesize that $\hat{E}^\star$ is more robust and accurate than $E$. In this section we explore these two questions, whether the model is realistic and additionally if it is able to provide more accurate estimations to the computational effort.

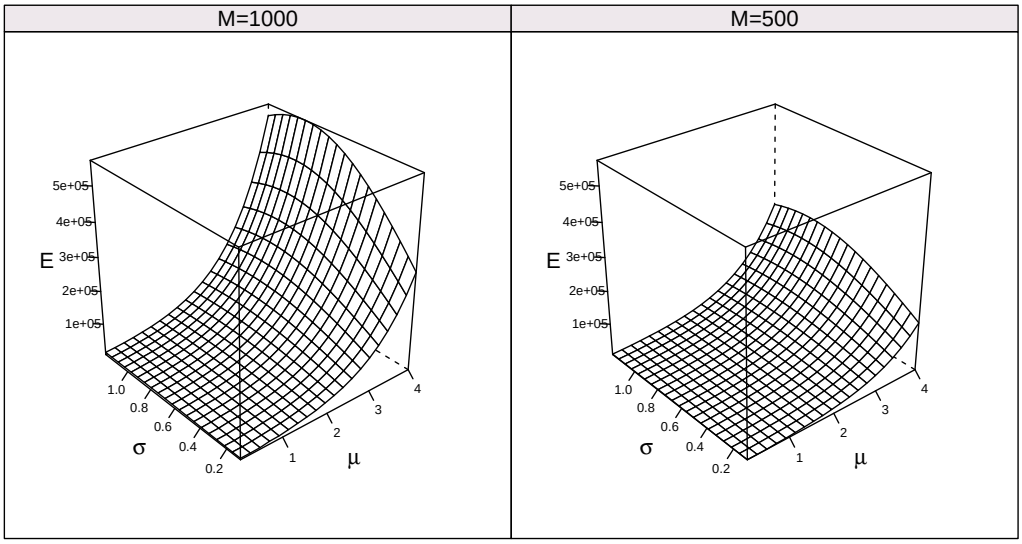


Figure 6.8: Graphical representation of the analitycal model of computational effort as a function of $\mu$, $\sigma$ for population size M=$\{500, 1000\}$, and SR=$0.5$, as modeled in (6.19).

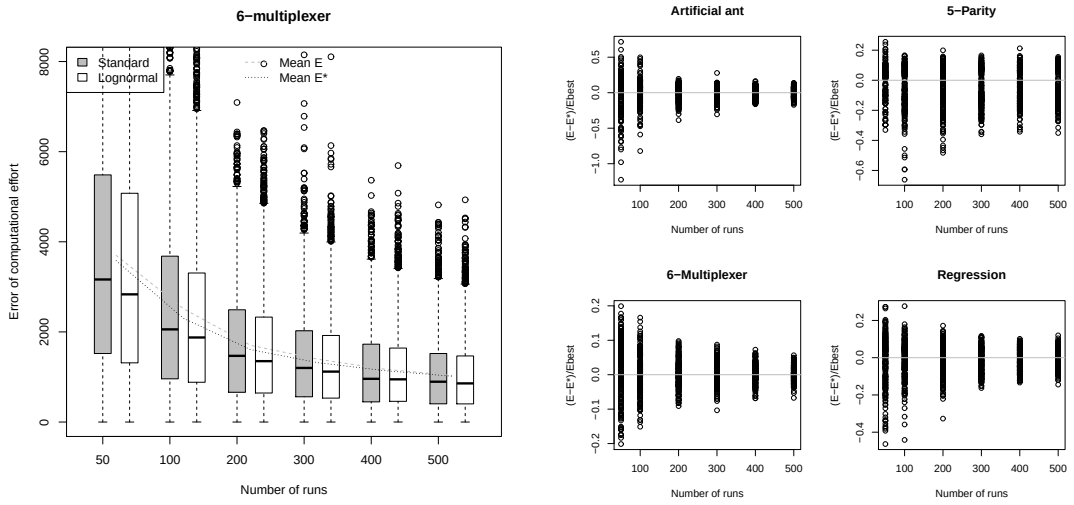


Figure 6.9: Left: Difference between $E$ and $E^\star$ with $E_{best}$ (low values are better) of the 6-multiplexer problem. Each box represents $2.000$ bootstrapped values of the error for different number of runs. No ceiling operator has been used. Right: Relative difference between $\hat{E}$ and $\hat{E}^\star$.

Table 6.2: Differences of $E$ and $E^\star$ with $E_{best}$ for different sample size. The table shows data for the regression problem, the same behaviour is found the other three problems under study. The computational effort has been calculated without ceiling operator regardless of the method.

| | $\lvert E_{best} - E\rvert$ | $sd\lvert E_{best} - E\rvert$ | $\lvert E_{best} - E^\star\rvert$ | $sd\lvert E_{best} - E^\star\rvert$ |
|---|---|---|---|---|
| 50 | 3627.9770 | 2562.9722 | 3482.4215 | 2910.1751 |
| 100 | 2585.3189 | 1961.6505 | 2416.7762 | 1900.6281 |
| 200 | 1634.6052 | 1264.5999 | 1572.8789 | 1204.3831 |
| 300 | 1368.5800 | 1013.5731 | 1300.1529 | 991.7907 |
| 400 | 1158.0780 | 888.1902 | 1118.1963 | 860.9419 |
| 500 | 1073.2470 | 790.5500 | 1012.5622 | 766.0491 |

In order to verify whether $\hat{E}^\star$ approximates better $E_{best}$ than $E$, we have performed the following experiment. We simulated $5,000$ experiments of the four problems sampling $n \in \{50, 100, 200, 300, 400, 500\}$ from the dataset. For each experiment, we calculated $\hat{E}_i$ and $\hat{E}_i^\star$ and then we depicted a boxplot with the differences $|\hat{E}_{best} - \hat{E}_i|$ and $|E_{best} - \hat{E}_i^\star|$. So, a good estimation of $E$ must generate boxes close to $0$. The result, for the case of the 6-multiplexer (the rest of the problems follow the same pattern), is shown in Figure 6.9 (left). The boxplot shows that the distribution of the error follows a skewed shape with a long tail represented by outsider points. It is clear that the greater is the sample size, the error associated to the measure is lower no matter which method is used. The improvement in quality of the lognormal-approximation used to estimate $E$ is not clear. While the standard deviation of the population is similar across the different number of runs, a slight difference in the median of the samples is found, more significant for low number of runs. It might be more clear in a numerical form rather than a graphical one, so it can also be read in Table 6.2.

In average terms, the modified method yields measures closer to the best estimation of $E$. Because of the significant variance of the measurements, as well as the small difference between the methods, we cannot claim that the precision of the measures is improved, however this fact repeated across all the number of runs and the high number resamples done in the experiment, $5,000$, lead us not to negate that. The mean values are represented in Figure 6.9 (left) with lines, and they show that the mean of the proposed method is lower than the original one for all the values of $n$.

The experiment previously reported served to compare the accuracy of $E$ and $E\star$ wrt to best estimation. Therefore, we only can conclude about their ability to yield accurace estimations of computational effort, but does not provide information about the accuracy of $\hat{E}^\star$. In order to check it out, we have performed another experiment, quite similar to the previous one. In this experiment we have simulated $200$ experiments, calculating $\hat{E}_i$ and $\hat{E}_i^\star$ for each pseudoexperiment. Then, we have depicted a scatterplot with the relative difference $(\hat{E}_i - \hat{E}_i^\star)/\hat{E}_{best}$ versus $n$ in Figure 6.9 (right). It can be seen that the relative difference is very small in the four problem instances. Even with a low number of runs ($50$), our model seems to be quite accurace, with a maximum relative error of 1%.

With these data in mind, we cannot state that the lognormal approximation to calculate

$E$ is more precise than the standard one using the same number of runs. Nonetheless, we can take the opposite view, the lognormal approximation, at least, does not seem to be less reliable than the standard method. This conclusion, in the context of this dissertation, is important, since it justifies that $E^\star$ is a reasonable model of $E$, and therefore, we can use it to characterize the error of $\hat{E}$.

### 6.6.3 Using the analytical model of $E$ to characterize its estimation error

Once we have an analytical model of the computational effort, we can address the main objective of this PhD thesis: characterize its error in order to determine the reliability of $E$. Our model is a function of three variables, which are SR, $\mu$ and $\sigma$, hence expressing $E$ in a fuctional form $E(SR, \mu, \sigma)$. The objective of this section is to analyze how an error in the estimation of any of its parameters affects the quality of the estimation. In order words, we want to determine

$$\Delta E = \left|\frac{\partial E}{\partial SR}\right| \Delta SR + \left|\frac{\partial E}{\partial \mu}\right| \Delta\mu + \left|\frac{\partial E}{\partial \sigma}\right| \Delta\sigma$$

The effects of the estimation error of SR has been partially studied in chapter 4, so we exclude it. However, the way that the estimation of $\mu$ and $\sigma$ can affect the reliability of $E$ is still unknown, so, in the following we focus our investigation to the study of these factors.

Given the analytical complexity of the model, and the lack of necessity of being strict, we simplify the problem using a numerical approach. A rude, but reasonable way, to estimate the relative error $\Delta E^{\%}$ numerically is just observing the difference of the exact computational effort and the computational effort calculated with an error in the estimation of its parameters

$$\Delta E^{\%} = \frac{E(SR, \mu, \sigma) - E(SR, \mu + \Delta\mu, \sigma + \Delta\sigma)}{E(SR, \mu, \sigma)}$$

The problem here is which values $\Delta\mu$ and $\Delta\sigma$ should be used. This problem is similar to the previous characterization of the estimation error of a probability, and can be addressed using the same strategy, which is relating error and CIs. In this case, there are two variables involved, and thus from a geometrical perspective, their estimation define an uncertainty surface whose boundaries would be approximated by the CIs of $\hat{\mu}$ and $\hat{\sigma}$.

So, the problem of determining the uncertainty region of the estimation of $E$, and thefore $\Delta\mu$ and $\Delta\sigma$, becomes a problem of computing lognormal CIs. But this is a problem that can be solved using the relationship between the normal and lognormal distributions, which is well known [150], and is given by the expression

$$X \sim N(\mu, \sigma) \Rightarrow e^X \sim LN(\mu_L, \sigma_L)$$
$$X \sim LN(\mu_L, \sigma_L) \Rightarrow \ln(X) \sim N(\mu, \sigma)$$

This relation is handy because provides a way to apply all the normal statistics to lognormal distributions, including normal CIs. It is well known that the CI of the mean $[\mu^-, \mu^+]$, given an unknown $\sigma$, is

$$[\mu^-, \mu^+] = \left[\hat{\mu} - t_{\alpha/2, n-1}\frac{\hat{\sigma}}{\sqrt{(n)}}, \hat{\mu} + t_{\alpha/2, n-1}\frac{\hat{\sigma}}{\sqrt{(n)}}\right] \tag{6.20}$$

where $t_{\alpha/2,n-1}$ is the upper $(1-\alpha)/2$ critical value for the Students' t distribution with $(n-1)$ degrees of freedom, and $n$ the number of samples (or runs in the context of EC). Similarly, the normal CI $[\sigma^-, \sigma^+]$ when the mean is unknown is given by

$$[\sigma^-, \sigma^+] = \left[ \sqrt{\frac{(n-1)\hat{\sigma}^2}{\chi_{\alpha/2,n-1}}}, \sqrt{\frac{(n-1)\hat{\sigma}^2}{\chi_{1-\alpha/2,n-1}}} \right] \tag{6.21}$$

where $\chi_{\alpha/2,n-1}$ and $\chi_{1-\alpha/2,n-1}$ are the upper and lower critical valuess for the $\chi$-squared distribution with $(n-1)$ degrees of freedom. The CIs defined in (6.20) and (6.21) provides the limits of the uncertainty region. Then, for each point $(\mu, \sigma)$, we determine its uncertainty region and estimate the relative error as

$$\Delta E^{\%} = \frac{\max(E(SR, \mu, \sigma) - E(SR, \mu', \sigma'))}{E(SR, \mu, \sigma)} \tag{6.22}$$

where $(SR, \mu, \sigma)$ is the point in the domain of $E$ where the error is being evaluated, and $(SR, \mu', \sigma')$ is a point contained in the surface defined by $(SR, \mu, \sigma)$ and $\sigma^- < \sigma < \sigma^+$ and $\mu^- < \mu < \mu^+$ that maximizes the difference $E(SR, \mu, \sigma) - \min(E(SR, \mu', \sigma')$. A graphical representation of $\Delta E^{\%}$ with confidence level $95\%$ is given in Figure 6.10. The figure plots the maximum expected relative error calculated with confidence level $0.95$ and $SR = 0.5$ for a domain that covers the parameter values reported in Table 5.3. Other values of SR were tried, however no significant differences were found.

Firstly, in Figure 6.10 we observe that the expected error is quite high when there is a low number of samples, but it decreases rapidly with $n$, additionally, increasing $n$ has as side effect a flatter error surface, and thus, the behaviour of the error seems to be more homogeneus when a large number of runs are used. Secondly, Figure 6.10 shows that the effect of $\mu$ is pretty moderate, and only in low values of this parameter, we can observe a dependence of the relative error with $\mu$ in form of smooth oscillations. The relative error depends much more on the dispersion of the run-time to success in a non trivial way. When $\sigma$ is low, the relative error seems to stay low, but as it is increased, the error also increases.

In this section we have developed a characterization of the relative estimation error of $E$. The model predicts high values of error associated to the measurement of $E$ in typical algorithms and experimental designs with $n = 30$. It also showed that the error remains almost constant with the mean run-time, while it is sensible to algorithms that exhibit high variances of the run-time to success. The model that we have proposed relies in several assumptions and simplifications that could reduce its reliability. For this reason, we complement the almost theoretical approach previously done, in order to verity its accuracy through experimentation.

## 6.7 Experimental analysis of Koza's performance measures

This chapter has approximated to problem of the reliability of the computational effort from almost a theoretical perspective. Ecen though it used a model of success probability that required some empirical observations. From our point of view, ideally theory and experimentation should complement each other in a double way: theory should be able to inspire

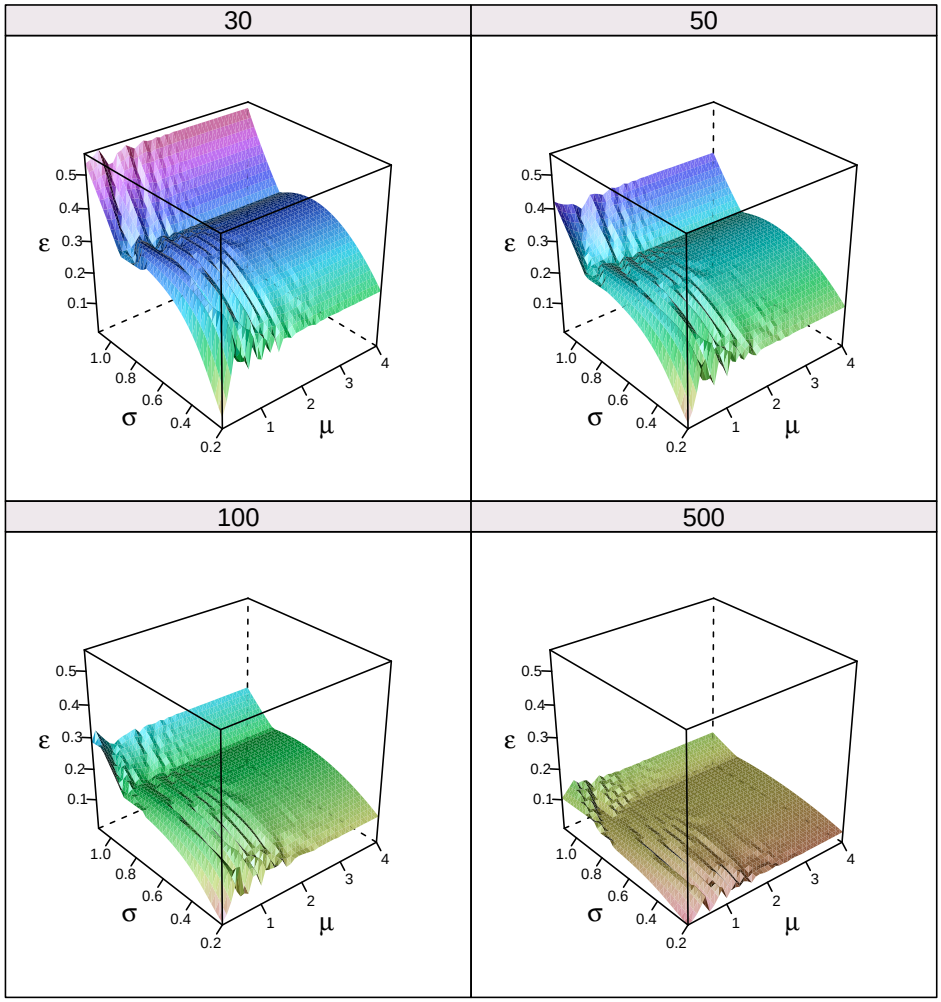


Figure 6.10: Maximum expected estimation error of $E$ as function of $\mu$, $\sigma$ and $n$, as modeled by (6.22). The number of runs takes values $n \in \{30, 50, 100, 500\}$. The SR is in all the cases $0.5$, different SR values do not yield remarkable differences.

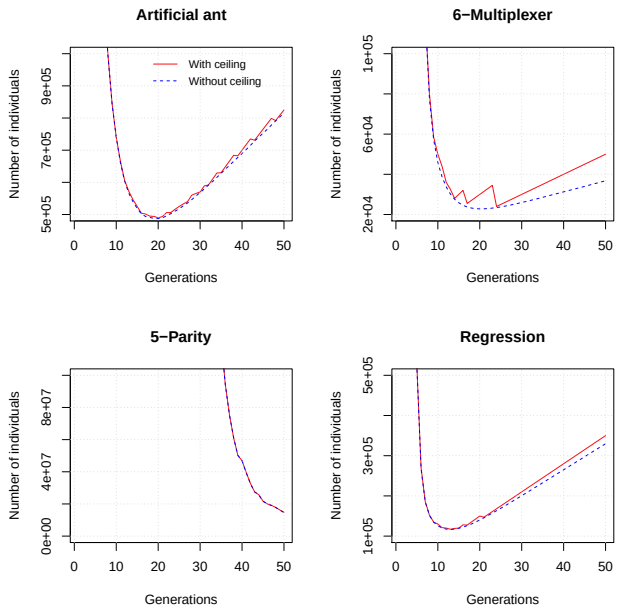


Figure 6.11: Comparison between the number of individuals to be processed computed using the ceiling operator (solid line) and not using it (dashed line). The curves have been calculated using all the samples in the dataset with $z = 0.99$.

new experiments, and, analogously, experimentation might be able to suggest new theories and verify them. Following this philosophy, in this section we try to verify the theoretical results obtained so far.

To be consistent with the theoretical work previous reported, we follow the same structure. We divide the problem into two assessing, on the one hand, the accuracy of $\hat{I}(M, i, z)$, followed, on the other hand, by an analysis of $\hat{E}$. In both cases, we consider the two factors that we have been studing along this chapter: the ceiling operator and the estimation error.

### 6.7.1 Accuracy of $\hat{I}(M, i, z)$

Before studing the reliability of $E$, we study the reliability of its main component, the number of individuals to be processed, or simply $I(M, i, z)$. We begin the empirical study looking at the effect of the ceiling operator in the estimation of $\hat{I}(M, i, z)$.

#### 6.7.1.1 Ceiling error of $\hat{I}(M, i, z)$

Firstly, we analyze the effect of the ceiling operator just plotting $\hat{I}^{best}(M, i, z)$ and $\hat{I}_c^{best}(M, i, z)$ in Figure 6.11. The most obvious difference is the sawtooth shape that $\hat{I}^{best}(M, i, z)$ has in some problem domains, such as the multiplexer. This shape is also found in the rest of the problems, nonetheless in different magnitude. In the case of the parity problem it seems that there are no discontinuities, however there are, but they are so small that only a zoom over the figure shows it. In any case, $\hat{I}^{best}(M, i, z)$ is strictly higher than $\hat{I}_c^{best}(M, i, z)$,

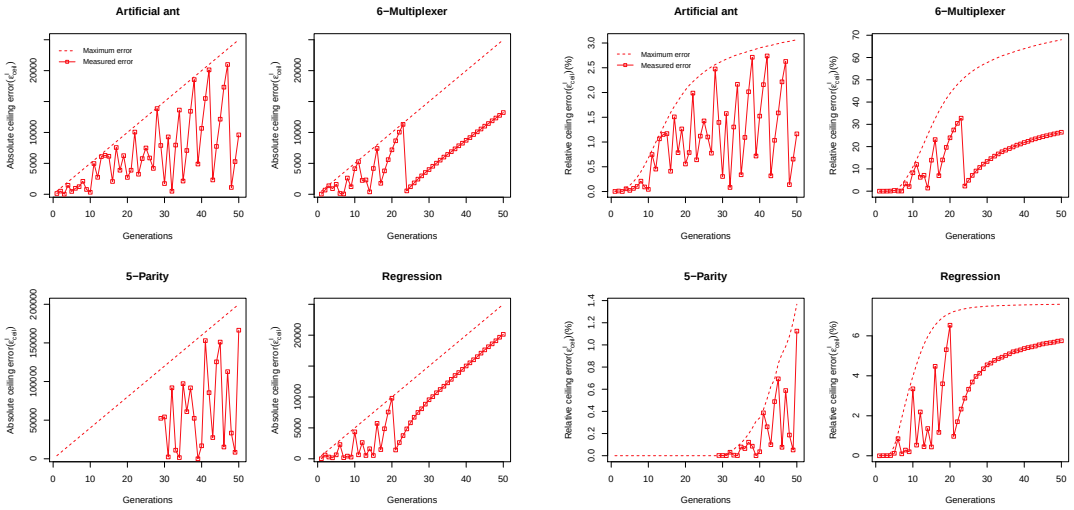


Figure 6.12: Comparison of theoretical and experimental ceiling error measures with all the runs. Error is reported in absolute (left) and relative (right) values.

so, as Christensen reported, the ceiling error is biased and tends to increase the value of $I^{best}(M,i,z)$.

Interestingly, there seems to be a correlation between the problem difficulty and the magnitude of the discontinuity; the ceiling operator introduces more discontinuities in the multiplexer problem ($\hat{P}_{best}(M,G) = 0.96$), followed by the regression ($\hat{P}_{best}(M,G) = 0.29$), artificial ant ($\hat{P}_{best}(M,G) = 0.13$) and finally the parity problem ($\hat{P}_{best}(M,G) = 0.06$). This experiment confirms the relationship between the ceiling error and the problem difficulty found by Christensen and Oppacher using a synthetic expression of $P(M,i)$ [54]. Experiments show that measuring $I(M,i,z)$ in easy problems tends to have more ceiling error than in hard problems. This fact is consistent with the theoretical work done in section 6.4.1.

The difference between $\hat{I}^{best}(M,i,z)$ and $\hat{I}_c^{best}(M,i,z)$ is better illustrated in the Figure 6.12 (left). This figure shows the difference $\hat{I}^{best}(M,i,z) - \hat{I}_c^{best}(M,i,z)$ with the maximum theoretical ceiling error set by (6.8). It can be seen that the theory describes very well the maximum error induced by the ceiling operator, this is particularly clear in the case of the artificial ant. The relationship between the ceiling error and the problem difficulty is clear looking at the relative ceiling error depicted in Figure 6.12 (right), where an easy problem such as the 6-multiplexer achieves a ceiling error up to 30%, when problems with a low success probability get much lower estimation error. For instance, the artificial presents at most an estimation error around 2.7%.

Despite the potentially high impact that the ceiling operator might have in the estimation, there is an easy solution, just removing the operator. Koza introduced this operator to reflect that it is not possible to carry out a fractional number of experiments [136, Chapter 4], however it is actually not supposed to be interpreted physically, so the ceiling error can be removed without any evident drawback. Nonetheless, the another source of variability under study, the estimation error, is intrinsic to the measure and thus cannot be removed.

#### 6.7.1.2 Estimation error of $\hat{I}(M,i,z)$

If we look in more detail (6.3), we can identify two fixed parameters, $M$ and $z$, and one independent variable, $i$. All these values are known, and thus they do not generate uncertainty. Usually, the only element in (6.3) that is not perfectly known is $P(M,i)$, that is an unknown probability and must be estimated empirically. The error associated to the estimation of $P(M,i)$ is actually the only true source of error since this is the only element in (6.3) that introduces uncertainty.

$\hat{P}(M,i)$ is the estimation of a probability, and, if we do not consider its variation in time, this probability in a fixed generation $i_0$ can be described using a binomial distribution, which is a well known problem [44]. Irrespective of the problem under study, the quality of the estimation of any success probability only depends on the number of trials (or runs in our case) and the magnitude of the probability [44], this result let us limit our study to only those factors.

We begin investigating the influence of the number of runs with the following experiment. Given the four datasets, we have calculated $5,000$ values of $\hat{I}_c(M,i,z)$ using $n$ runs resampling with replacement from each dataset. The ceiling function is removed to isolate the effects of the estimation error. For each value of $\hat{I}_c(M,i,z)$, its distance to $\hat{I}_c^{best}(M,i,z)$ has been calculated using the following formula

$$\overline{\xi} = \sum_{j=0}^{R} \sum_{i=0}^{G} \frac{\hat{I}_c^{best}(M,i,z) - \hat{I}_c^{j}(M,i,z)}{R} \tag{6.23}$$

where $\overline{\xi}$ is the statistic that measures the average distance between $\hat{I}_c^{best}(M,i,z)$ and $\hat{I}_c^{j}(M,i,z)$, which is the $j^{th}$ curve of the number of individuals to be processed. $R$ is the number of pseudo experiments. All the experiments were carried out with $R = 5,000$ and $G = 50$. Of course, it is an error measure and therefore, low values mean good estimations.

The boxplots of the estimation error calculated using the method described earlier are depicted in Figure 6.13. A glance to this figure clearly suggests a strong relationship between the number of runs and the average estimation error, more runs yield better estimations of $I(M,i,z)$. The estimation error of the 5-parity problem is not shown because it was found that the low number of generations where $I(M,i,z)$ is defined (see Figure 6.1) induced an erratic distance behaviour.

Experimentation with the other factor under study, the accumulated success probability, is more tricky. $P(M,i)$ is not an independent variable, but a dependent one and, unless we use a synthetic $P(M,i)$, we cannot manipulate it to carry out the experiment. Additionally, $P(M,i)$ is a function rather than a scalar. These two facts difficult experimentation, however, we can still perform an experiment to observe the behaviour of the estimation error for different values of the accumulated success probability. For each problem domain, we have run $200$ pseudoexperiments with $n = 100$ following the same procedure described above, but we have done a different manipulation of the data. Instead of measure how close is $\hat{I}_c^{best}(M,i,z)$ from $\hat{I}_c(M,i,z)$, we have stored the tuple $(P(M,i), \varepsilon_{est}(i))$, where $i = 1, ..., G$ and

$$\varepsilon_{est}(i) = 100 \frac{\hat{I}_c^{best}(M,i,z) - \hat{I}_c(M,i,z)}{\hat{I}_c^{best}(M,i,z)}$$

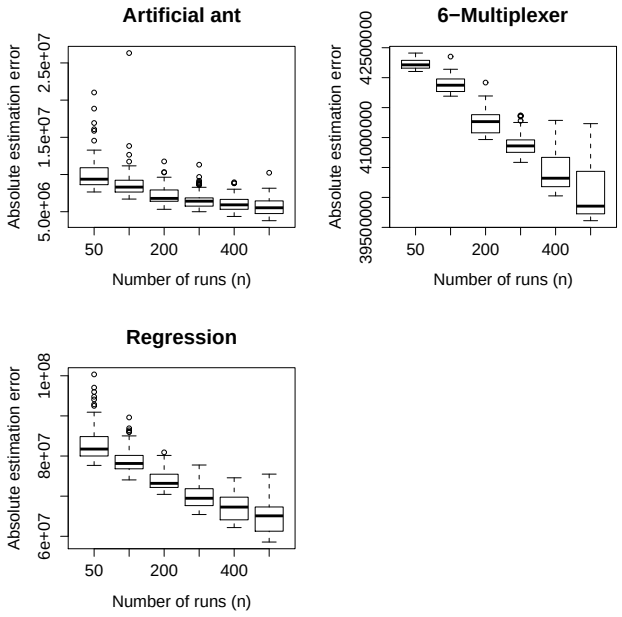


Figure 6.13: Boxplot of the absolute estimation error of $\hat{I}(M, i, z)$ with several values of number of runs. Each box represents the sum of the average estimation error of $5.000$ pseudoexperiments.

is the relative estimation error. In this way we obtain $G = 50$ tuples from each pseudorun, and we used 200, so there are $10,000$ tuples in each problem domain.

The tuples that we have obtained are shown in the scatterplot depicted in Figure 6.14. This figure shows a surprising behaviour of the estimation error: it is not symmetrical and it is biased. Overestimating $P(M, i)$ yields an underestimation of $I(M, i, z)$, on the contrary, an overestimation of $P(M, i)$ generates an underestimation of $I(M, i, z)$. Figure 6.14 shows that the effects of overestimating or underestimating $P(M, i)$ are not the same. An overestimation of $P(M, i)$ induces a higher error in $I(M, i, z)$ than a underestimation, it is specially notorious in the case of the artificial ant and the 5-parity problems. This asymmetry varies with the success probability, while the minimum error tends to reduce with the probability, the maximum error is almost constant. In any case, there is an asymptotic behaviour of the estimation error with very low success probability that makes the estimation highly imprecise in that region.

The magnitude of the maximum estimation error depends on the success probability. Low probabilities yield higher estimation error and higher success probabilities tend to generate less estimation error. Nonetheless the error is biased in the end of the execution of the algorithm (higher success rates), with the only exception of the multiplexer, which is the only one that achieve a success rate close to 1. It leads us to conjecture that high success probabilities have associated higher estimation error, however we feel unable to claim it with the evidence shown, it should be confirmed by further research. In any case, the magnitude of the bias seems to be rather significant in almost all the cases, around $30\%$ and $50\%$, with the exception of the 6-multiplexer. We should remark that this experiment used $100$ runs, which

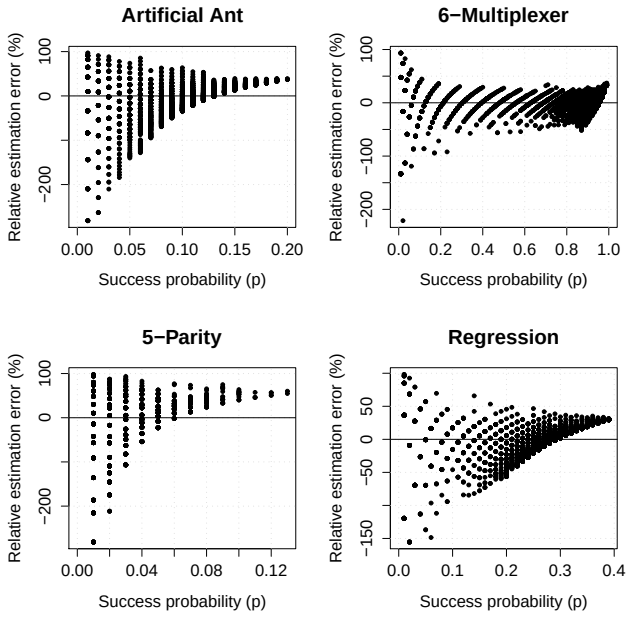


Figure 6.14: Scatterplot of the relative estimation error of $I(M, i, z)$ with several values of success probability. 200 pseudoexperiments with $n = 100$ were carried out for each problem domain.

is a relatively high number of runs; it is quite easy to find literature that reports experiments with fewer number of runs, so we can expect that the estimation error in those experiments were higher.

In average, the relative estimation error is notable and the estimator is biased significantly, depending on the problem domain. Nonetheless, these error might be, or not, significant when the computational effort is calculated, which is the objective of the next subsection.

### 6.7.2 Accuracy of $\hat{E}$

Common sense suggests that a good estimation of $I(M, i, z)$ should also yield a good estimation of the computational effort; this apparent correlation should link the factors of $I(M, i, z)$ with the factors of $E$. However, common sense might fail, therefore we have performed some experiments to verify this hypothesis. We should point out that in this section we only study one factor, the number of runs. There are reasons to think that the magnitude of the accumulated success probability plays an important role, however we must face that this probability is not fixed with the generation time and it is not an independent variable. Moreover, the variation of $P(M, i)$ plays an essential role in the measurement of the computational effort, and it is not possible to treat it as a punctual estimator, like we did in the previous section. For these reasons, in the following, this factor is excluded from the study.

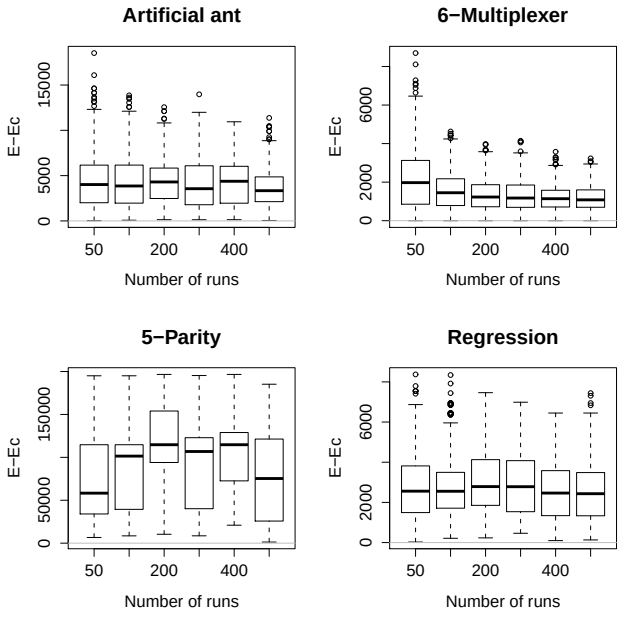


Figure 6.15: Absolute ceiling error of computational effort with several number of runs. Each box represents $2,000$ pseudoexperiments.

#### 6.7.2.1 Ceiling error of $\hat{E}$

Firstly it is worth to compare the computational effort with and without ceiling operator when it is calculated using all the samples. These values, as well as their absolute and relative difference, can be found in Table 6.1. We found earlier that easy problems -those with high success probability- generated more ceiling error in the estimation of $I(M,i,z)$. Table 6.1 shows that our experiments partially verify this behaviour in the estimation of $E$. The easiest problem, the 6-multiplexer ($\hat{P}_{best}(M,G) = 0.96$), generated the biggest difference between $\hat{E}^{best}$ and $\hat{E}_c^{best}$, $4.98\%$, while the rest of the domains achieve intermediate values of ceiling error: the artificial ant ($\hat{P}_{best}(M,G) = 0.13$) with a difference of $0.5\%$, the 5-parity ($\hat{P}_{best}(M,G) = 0.06$) with $1.13\%$ and finally the regression problem ($\hat{P}_{best}(M,G) = 0.29$) with $0.49\%$.

There is no direct correlation between problem difficulty and ceiling error when estimating $E$. There may be two possible explanations behind this fact. First, the ceiling operator introduces discontinuities in $I(M,i,z)$ that might increase variance when the minimum is calculated. From another perspective, we observe that the 5-parity is the hardest problem but it has the second highest ceiling error. There is an important issue with this problem domain, as can be seen in Figure 6.1: the number of generations given to the algorithm is too scarce, so it could have affected the result of this experiment. So far, it seems to be a tight correlation between the success probability of a problem and the ceiling error associated to $\hat{E}$. As we did earlier, we pass to study whether the number of runs influences the ceiling error.

Figure 6.15 shows a boxplot that represents the difference $\hat{E} - \hat{E}_c$ of $2,000$ pseudoexperiments calculated with different values of $n$. The use of $\hat{E}^{best}$ has been avoided to isolate the

Table 6.3: Analysis of variance for six levels of factor $n$, the independent variable is the square root of the difference $E_c - E$. Residuals of problems marked with * did not pass the normality test. P-values with significance ($\alpha = 0.01$) are marked in bold.

| Problem | df | Sum. sq. | Mean sq. | F-value | p-value |
|---|---|---|---|---|---|
| Artificial ant | 5 | 2445 | 489.03 | 0.9689 | 0.437 |
| 6-Multiplexer | 5 | 5145 | 1029 | 5.1462 | **0.0001529*** |
| 5-Parity | 5 | 236380 | 47276 | 4.9595 | **0.0002247*** |
| Regression | 5 | 4349 | 869.79 | 2.9602 | 0.01266 |

ceiling error from the estimation error. We first observe that the difference is always positive, meaning that $E > E_c$, which is not surprising because the ceiling operator always increases its argument, unless it were an integer, which is rather unlikely. No notable differences in the mean value of the difference $\hat{E} - \hat{E}_c$ are appreciated, only when $n$ is small, around $50$ runs, the tail of the distribution seems to be longer, with more outsiders, but the median, as well as the first and third quantiles, remains almost constant, regardless of the number of runs.

This result is confirmed with a one-way ANOVA test, whose result is shown in Table 6.3. The ANOVA was calculated for six levels of $n$ ($50$, $100$, $200$, $300$, $400$ and $500$) using the square root of $\hat{E} - \hat{E}_c$ as independent variable. Using $50$ pseudoexperiments for each level, two problems (multiplexer and parity) yielded statistical significance with $\alpha = 0.01$ while two did not (artificial ant and regression). However, the residuals of the multiplexer and the parity problems did not pass the normality test, and therefore we cannot accept their test as valid. The residuals of the other two problems did pass the normality test, which are the two that did not found differences, so, with this evidence, we conclude that the number of runs does not affect the ceiling error when estimating computational effort.

Experiments shown in this subsection were designed to avoid the effects of the estimation error, which is just the factor that we move forward to study.

#### 6.7.2.2 Estimation error of $\hat{E}$

Finally, we study the effects of the estimation error. This study follows a procedure similar to the one used previously. Given the datasets of the four selected problem domains, $100$ experiments were simulated resampling $n$ runs with replacement from the datasets. For each simulated experiment, the error between the estimation and the best estimation of computational effort was calculated. Two methods to calculate computational effort were used, using the ceiling operator and not. In this way, we are able to measure the estimation error as well as compare both methods of calculating computational effort, so the statistic of relative estimation error is given by

$$\varepsilon_{est}^{E}(\%) = \frac{E_{best} - E}{E_{best}};\ \varepsilon_{est}^{E_c}(\%) = \frac{E_c^{best} - E_c}{E_c^{best}}$$

The variation of the estimation error with $n$ is shown in Figure 6.16. It shows some interesting behaviors. Probably, the most important one from a practical point of view is the

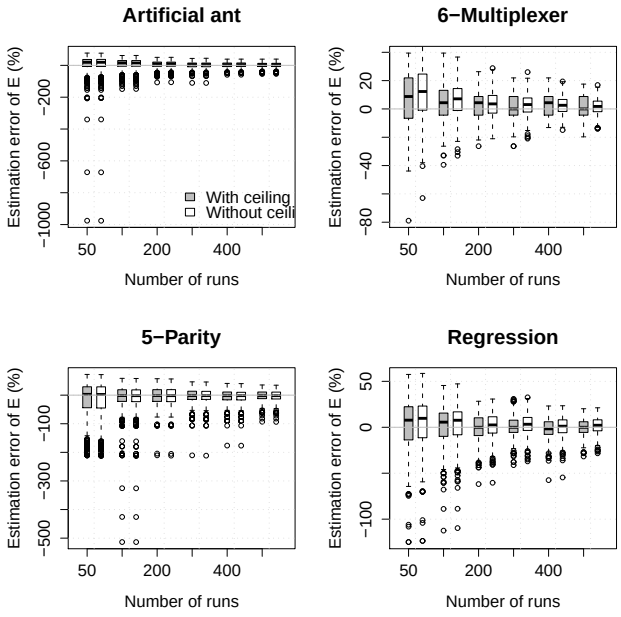


Figure 6.16: Estimation error of computational effort with several values of number of runs. Each box represents $100$ pseudoexperiments.

high relative estimation error found in our experiments. Depending on the problem, when the number of runs is not too high, an estimation error of computational effort up to $50\%$ is found. Error decreases rapidly with the number of runs, however there is a point that a small reduction of the error requires a very remarkable increment of the number of runs. Depending on the context, incrementing the number of runs might not pay off.

Another interesting property that Figure 6.16 shows is the asymmetry of the estimation error. It was previously shown that estimation error of $I(M, i, z)$ is asymmetrical and we can observe now that this behaviour is transferred to the estimation error of $E$. The maximum overestimation of $E$ is bounded and it tends to reduce its value as $n$ increases. Unfortunately, when $E$ is underestimated, it tends to produce much higher errors, nonetheless this difference tends to disappear when the number of runs is increased. Finally, the ceiling operator does not seem to influence the estimation error, the distribution of the estimation error with and without ceiling operator is similar, with the only exception of the 6-multiplexer, which is also the most sensitive problem to the ceiling operator.

Although the variation of the estimation error shown in Figure 6.16 is rather clear, it is better support this conclusion with a statistical test. We performed a one-way ANOVA of the square root of the estimation error for the six levels of $n$ previously shown, the result can be seen in Table 6.4. The test found differences in the levels of the factor for the four problems using a significance level $\alpha = 0.01$, however one problem, the 5-parity, did not pass the normality test of its residues.

Table 6.4: Analysis of variance for six levels of factor $n$, the independent variable is the square root of the estimation error of $E_c$. Residuals of problems marked with * did not pass the normality test. P-values with significance ($\alpha = 0.01$) are marked in bold.

| Problem | df | Sum sq | Mean sq | F value | p-value |
|---|---|---|---|---|---|
| Artificial Ant | 5 | 126.44 | 25.2874 | 12.579 | **1.035e-10** |
| 6-Multiplexer | 5 | 95.09 | 19.0180 | 14.654 | **3.272e-12** |
| 5-Parity | 5 | 98.72 | 19.7441 | 8.4623 | **4.308e-07*** |
| Regression | 5 | 106.12 | 21.2248 | 12.264 | **3.414e-10** |

## 6.8 Conclusions

Koza's performance measures have been widely used in the GP literature. It makes surprising the lack of attention that the understanding of this statistic has attracted. In this chapter we have tried to provide some clues to reduce the gap between the importance of Koza's performance measures and the knowledge about its behaviour and reliability.

In short, we can identify one source of variability, the ceiling operator, and one source of randomness, the estimation of success probability, associated to the measurement of the number of individuals to be processed. The ceiling operator introduces a maximum relative error that is arbitrary high depending on the success probability. High success probabilities generate high error values induced by this operator, meanwhile small values of $P(M, i)$ are associated to small ceiling errors. From the perspective of absolute value of the maximum ceiling error, it is linearly limited by the product of the population size and the generation number. It is possible to remove the operator without significant drawbacks. So, results reported in this chapter recommend, in the same line than some previous authors, not using the ceiling error when calculating $I(M, i, z)$.

The only source of randomness in the measurement of $I(M, i, z)$ is introduced by the estimation of the success probability. An estimation of this error can be done using CIs. Basically, the quality of the estimation of a success probability depends on two factors: the number of runs and the value of the probability. The worst scenario is estimating a success probability close to $0$ or $1$ with a low number of runs, in that case the estimations are very unrealible. This is just the scenario found in early stages of the EA, and there is only one method to improve the reliability of the measure: increasing the number of runs. In case there were a high number of runs, and a low success probability, successes can be modeled using a Poisson distribution instead of a binomial. The analytical approximation to characterize the error associated to the estimation of the success probability was supported by experiments. These experiments validated the analytical models.

As a final observation, measures studied in this chapter are not the only options to gather information about EA behaviour. A comparative study of performance measures would provide useful information about their behaviour, advantages and disadvantages, even more, it could provide clues about which measure, when, and how, should be used to achieve better experimentation in EC. Finally, we should emphasize that performance is only a restricted view of all the picture. To fully understand what happens within an EA, other measures should be taken into account.

# Chapter 7

# Conclusions and future work

*Un soneto me manda hacer Violante,*
*que en mi vida me he visto en tal aprieto;*
*catorce versos dicen que es soneto:*
*burla burlando van los tres delante.*

*Yo pensé que no hallara consonante*
*y estoy a la mitad de otro cuarteto;*
*mas si me veo en el primer terceto*
*no hay cosa en los cuartetos que me espante.*

*Por el primer terceto voy entrando*
*y parece que entré con pie derecho,*
*pues fin con este verso le voy dando.*

*Ya estoy en el segundo, y aun sospecho*
*que voy los trece versos acabando;*
*contad si son catorce, y está hecho.*
*Lope de Vega*

This chapter summarizes the main conclusions of the dissertation and some research lines that remain open are also described.

## 7.1 Conclusions

The main goal of this dissertation has been to characterize the error associated to the measurement of Koza's computational effort. With this characterization, it is possible to draw an answer to the main research question that drives this PhD thesis, which is to determine

whether the computational effort is a reliable performance measure, or, on the contrary, it is not. Since the main research question is too wide and involves a collection of different issues, it was convenient to split it into five specific research questions. In the following, we review these specific questions, which were already presented in the introduction, and their answers are discussed under the light of the evidence reported along this dissertation.

- **Q1: Which factors influence the reliability of the computational effort?**
  Computational effort, as defined by John Koza, only contains *two sources of variability*, one deterministic and another stochastic (section 6.4). This claim is a direct consequence of the definition of the computational effort. The deterministic source of variability is the *ceiling operator*. It removes part of the information introducing a deterministic bias in the results depending on the magnitude of the measure (section 6.4.1). The only source of randomness is the *estimation of the success probability*, this source of error cannot be removed, and thus, it is the main source of uncertainty.

  The basic measure that determines the quality of the estimation of the computational effort is the success probability, which is a probability function that depends on the time. In order to characterize it from a statistical point of view, it is useful to decompose this problem into two: the success probability when time is fixed (static estimation) and the variation with the time of the success probability (dynamic estimation) (section 5.4.1). This decomposition relates to the two questions that the success probability answers: *How* likely is it to find the solution and *when* is the solution likely to be found.

- **Q2: Which statistical properties the static estimation of the success probability has?**
  If time is fixed, an EA may be described as a simple Bernoulli process, i.e., an experiment with only two possible outcomes, that we name "success" or "failure". By definition, the number of success (and therefore the success rate) in a Bernoulli process is a binomial random variable. Empirical and theoretical evidences reported in sections 4.3 and 4.6 support this claim.

  The statistical properties of the success rate in any EA are related to the binomial distribution, and therefore its quality depends only on two factors, the *number of trials* and the *estimated probability* (section 4.4). A low number of trials and extreme probabilities close to $0$ or $1$ set the ideal conditions to generate bad estimations. It follows that the quality of the estimation does not depends directly on the algorithm internals, but only indirectly through the value of the success probability. Easy and hard problems would yield probabilities close to the boundaries $0$ and $1$, and thus the quality of the estimation get worse (section 4.5. In these cases, the success rate should be better approximated using alternative distributions (section 4.5.5).

  The binomiality of the static estimation of the success probability opens the opportunity to use binomial statistics into EC. Perhaps, one of the most interesting statistical tools, and indeed a tool that we needed in order to accomplish the main research goal, is confidence intervals (CIs). They provide a region where the success rate is likely to be contained with a certain nominal probability. Many binomial CI methods have

been reported in the statistical literature, but we analyzed four methods under the perspective of EC: Standard, Agresti-Coull, "exact" and Wilson.

The quality of a CI can be reported using two measures, coverage probability (CP) and confidence interval width (or CIW) (section 4.5). Experiments in section 4.6 showed that the CP and CIW of intervals calculated in EC follow the same behaviour that has been reported in the statistical literature. After an analysis of the four binomial CI methods, we concluded that Wilson has some properties that make it better as a general purpose binomial CI method (4.5.5). Under certain special contexts, "exact" and Agresti-Coull methods would be a better choice, while the standard method showed very poor performance, and its usage is not advisable in any case.

- **Q3: Which statistical properties the dynamic estimation of the success probability has?**
  There is not a definitive answer to this question yet, however we can outline an answer. The dynamic estimation of the success probability is closely related to the run-time analysis of the EAs, which is a problem widely studied in the context of Metaheuristics and Stochastic Local Search, but, to the author's knowledge, it has not been addressed before in GP.

  The term run-time to success was introduced in section 5.2 as a new tool to analyze the run-time of EAs, and it was empirically studied in GP in order to find a statistical model able to describe it (section 5.2.1). We found that, in general, the run-time to success is a lognormal random variable. There are some remarkable exceptions, notoriously, difficult boolean problems. In these cases, if the left tail of the distribution is removed, the remaining samples fit a exponential distribution (section 5.2.2), suggesting that the search is performed without learning. If the tournament selection is replaced by a random selection, and thus any selective pressure has been removed, the resulting distribution of the run-time to success fits nicely to a Weibull distribution (section 5.2.3). So far, it seems reasonable to conclude that these three distributions play a role in the description of the run-time in GP, moreover, the literature suggests that this behaviour likely can be generalized to Metaheuristics and Stochastic Local Search algorithms (section 5.5).

  As a consequence of the run-time analysis performed, we concluded that the dynamic estimation of the success probability can be done estimating the parameters of a lognormal, Weibull or exponential distribution. In particular, we provided empirical data supporting this claim in case of algorithms with a lognormal run-time distribution (section 5.4.3). Thus, classical statistical methods to estimate the parameters of a distribution can be applied to solve this problem, including confidence intervals or maximum-likelihood, whose properties are well known in Statistics.

- **Q4: Can the success probability be analytically modeled?**
  Yes, at lesat, in four problem instances, as seen in section 5.4.3). On the one hand, the static success probability at the end of the run comes from a binomial distribution, and the well known maximum-likelihood method can be used (section 4.3). On the other hand, the dynamic estimation of the success probability can be deduced from the run-time behaviour of the algorithm (section 5.4.1). It was observed that the lognormal

distribution describes reasonably well the run-time distribution of tree-based GP, and thus it is a good candidate to be used in a model of success probability (section 5.4.2). With these considerations, an analytical model of success probability was proposed, and its accuracy tested experimentally, finding out that the model approximates well the success probability (section 5.4.3). The main limitation of the model come from the estimation of the success rate, needed by the model, while it shows to be robust to bad fits of the run-time distribution.

In addition to the previously described empirical approach, a theoretical model based on Discrete-Time Markov Chains of the convergence of an iterative stochastic search algorithm was proposed in section 5.3. Using this model, we demonstrated that the exponential run-time behaviour observed in tree-based GP is a general property of memoryless iterative stochastic search algorithms. If the algorithm is memoryless, its run-time to success is exponentially distributed. This theoretical result is consistent with the experimentation (section 5.2.2) and related literature (section 5.5), where only difficult problems, when the initialization phase is removed, have an exponential behaviour.

- **Q5: Does the run-time behaviour provide information about the algorithm?**
  This question is still open and requires further research. We have observed, and the literature supports it, that the resulting run-time distribution of the algorithm depends on the parameters and problem difficulty. However, the opposite deduction is not clear, given a certain run-time distribution, can we infer some knowledge about the algorithm? In case of an affirmative answer, it would open a simple new method to analyze stochastic search algorithms with a minor computational overhead.

Based on the previous answers to specific research questions Q1 to Q5, we were able to accomplish the main goal of this thesis, characterize the estimation error of the computational effort. Once we identified the two sources of uncertainty in the measurement of computational effort, the ceiling operator and the estimation of the success probability, we characterized the error induced by both.

It was analytically demostrated in section 6.4.1 that the *ceiling operator* introduces as error that is bounded by the product of the generation and the population size. In relative terms, this error is a non-linear function of the success probability. The maximum relative ceiling error grows non-linearly with the value of the success probability, up to a point where the success probability is higher than the parameter $z$ and the measure is no longer valid. *A straightforward solution to eliminate the ceiling error is just not using the ceiling operator*. Koza justified its use to represent that an algorithm only can be run an integer number of times. In practice, this measure is not used to estimate the number of runs needed in an experiment, but rather to estimate the amount of resources used to achieve a solution. Therefore, the ceiling operator does not provide a practical advantage while it introduces notable problems.

The second source of error comes from the *estimation of the success probability*, and is much more difficult to characterize. This error source is intrinsic to the measuring procedure and cannot be eliminated. In order to characterize it, we used the model of success probability developed to answer Q4 in section 5.4.2. This model depends on three parameters

that have to be estimated: Mean and variance of the run-time to success, and the success rate. In order to model the error associated to the measurement, we used confidence intervals to model the uncertainty. This model predicts (section 6.6.3) that the estimation error does not depend on the estimation of the mean, but it correlates with its dispersion in a non trivial way, but in general we can assume that higher variability of run-time to success involves higher errors in the estimation of the computational effort. These results were experimentally validated in section 6.7.

In addition to the previous conclusions, we can make some general comments. From an analytical point of view, Koza's computational effort has a serious problem, due to its non-linearity, small estimation errors of the success probability, under certain conditions, are amplified to a point that the measure is not reliable at all. Moreover, the computation of the metric involves a fundamental measure (the success probability), the population size, and a new parameter ($z$), increasing the complexity, and introducing the effects previously described.

It seems reasonable to ask why it introduces all this complexity, and which is the advantage of such increased complexity. Koza justified it as a way to take into consideration not only the time required to find the solution, but also the population size, which determines the resources wasted in the search. From our point of view, it is better measuring the number of evaluations used to achieve the solution, which provides, at least, the same information, without any of the drawbacks previously described. Another viable alternative would be reporting the success probability and the population size, in this way we avoid the non-linear effects, providing a more reliable information about the algorithm.

For all these reasons, and as a general conclusion of the dissertation based on the evidence reported in this memory, *we suggest not using Koza's computational effort*. In our opinion, it is unnecessarily complex and unreliable. Based on the Occam's razor principle, we suggest using simple measures such as the success probability or the average number of evaluations.

## 7.2 Future work

There are some topics related to the reliability of the computational effort that have not been addressed in this dissertation. For instance, the iterations between the success rate and the other two parameters of the model have not been studied. In addition, in chapter 5 we obtained three statistical distributions that could be used in our model, but only one of them, the lognormal, was included in the study in order to represent the most general case. However, despite all these flaws, we think that this research line does not pay off: the main conclusion of this work is that the computational effort should not be used; probably the model of error can be enhanced, but it hardly would change the main concusion of the dissertation.

Nonetheless, along the way that we have followed to accomplish the main objective of the thesis, several new questions have arisen, opening new promising research lines. In the section 5.3, we proposed a model of run-time to success distribution based on Markov chains, and using this model we deduced the conditions that yield an exponential run-time to a solution. However, we did not explore this line enough to verify theoretically the conditions

that generate a lognormal or a Weibull distribution. Monte Carlo simulation seems to be a tool that could be used in order to justify the run-time behaviour observed in the experimentation. In particular, it would be interesting to analyze, given a certain run-time distribution, what could be known of the algorithm and problem at hand. The generality of the proposed model, and the related literature, suggest that the conclusions obtained in this way could be generalizable to a large number of algorithms, including Metaheuristics. Following this line, it would be interesting to extend this type of study to the population, and try to understand how it changes with time.

The run-time analysis that we have performed has used the generation as time unit. It was done motivated by our object of study, which is computational effort, however, it is not the most popular time measure in the EC community. There are strong reasons to hypothesize that the observations made so far relating to the generation-to-success can also be extended to any time unit. It would be interesting to check out if more popular time units, like the average number of evaluations to a solution, follow the same pattern. Linked to this, we plan to extend the experimental analysis in order to include other algorithms and problems, for instance, multiobjective algorithms and real world problems.

# Bibliography


[1] B. Adenso-Diaz and M. Laguna. Fine-Tuning of Algorithms Using Fractional Experimental Designs and Local Search. *Operations Research*, 54(1):99–114, 2006.

[2] A. Agresti and B. A. Coull. Approximate is Better than 'Exact' for Interval Estimation of Binomial Proportions. *The American Statistician*, 52:119–126, May 1998.

[3] R. Aler, J. M. Valls, D. Camacho, and A. Lopez. Programming Robosoccer Agents by Modeling Human Behavior. *Expert Systems with Applications*, 36(2):1850–1859, 2009.

[4] L. Altenberg. *Handbook of Evolutionary Computation*, volume 2, chapter NK fitness landscapes, pages B2.5:5–B2.7:10. IOP Publishing and Oxford University Press, Bristol and Oxford, 1997.

[5] P. J. Angeline. Adaptive and Self-Adaptive Evolutionary Computations. In *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163, 1995.

[6] P. J. Angeline. An Investigation Into the Sensitivity of Genetic Programming to the Frequency of Leaf Selection During Subtree Crossover. In *Proceedings of the First Annual Conference on Genetic Programming (GECCO 96)*, pages 21–29, Cambridge, MA, USA, 1996. MIT Press.

[7] P. J. Angeline. A Historical Perspective on the Evolution of Executable Structures. *Fundamenta Informaticae*, 35(1-4):179–195, 1998.

[8] T. Back, D. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., 1997.

[9] T. Back, U. Hammel, and H.-P. Schwefel. Evolutionary Computation: Comments on the History and Current State. *IEEE Transactions on Evolutionary Computation*, 1:3–17, 1997.

[10] W. Banzhaf. Genetic Programming for Pedestrians. In *Proceedings of the 5th International Conference on Genetic Algorithms (ICGA-93)*, page 628, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.

[11] R. Barr, B. Golden, J. Kelly, M. Resende, and W. Stewart. Designing and Reporting on Computational Experiments with Heuristic Methods. *Journal of Heuristics*, 1:9–32, 1995.

[12] R. Barr and B. Hickman. Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinions. *ORSA Journal on Computing*, 5:2–2, 1993.

[13] D. F. Barrero, D. Camacho, and M. D. R-Moreno. A Framework for Agent-Based Evaluation of Genetic Algorithms. In *Proceedings of the 3rd International Symposium on Intelligent Distributed Computing (IDC 2009)*, volume 237, pages 31–41, Ayia Napa, Cyprus, 13-14 October 2009. Springer-Verlag.

[14] D. F. Barrero, D. Camacho, and M. D. R-Moreno. *Data Mining and Multiagent Integration*, chapter Automatic Web Data Extraction based on Genetic Algorithms and Regular Expressions, pages 143–154. Springer-Verlag, University of Technology Sydney, Australia, July 2009.

[15] D. F. Barrero, D. Camacho, and M. D. R-Moreno. Confidence Intervals of Success Rates in Evolutionary Computation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*, pages 975–976, Portland, Oregon, USA, jul 2010.

[16] D. F. Barrero, B. Castaño, M. D. R-Moreno, and D. Camacho. Statistical Distribution of Generation-to-Success in GP: Application to Model Accumulated Success Probability. In *Proceedings of the 14th European Conference on Genetic Programming, (EuroGP 2011)*, volume 6621 of *LNCS*, pages 155–166, Turin, Italy, 27-29 Apr. 2011. Springer Verlag.

[17] D. F. Barrero, A. González-Pardo, D. Camacho, and M. D. R-Moreno. Distributed Parameter Tuning for Genetic Algorithms. *Computer Science and Information Systems*, 7(3):661–677, Jun 2010.

[18] D. F. Barrero, A. González-Pardo, M. D. R-Moreno, and D. Camacho. Variable Length-Based Genetic Representation to Automatically Evolve Wrappers. In *Proceedings of 8th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2010)*, volume 2, pages 371–379, Salamanca, Spain, 26-28 April 2010. Springer-Verlag.

[19] D. F. Barrero, M. R-Moreno, B. Castaño, and D. Camacho. An Empirical Study on the Accuracy of Computational Effort in Genetic Programming. In *Proceedings of the 2011 IEEE Congress on Evolutionary Computation (CEC 2011)*, pages 1169–1176, New Orleans, USA, 5-8 June 2011. IEEE Press.

[20] D. F. Barrero, M. D. R-Moreno, and D. Camacho. Adapting Searchy to Extract Data Using Evolved Wrappers. *Expert Systems with Applications*, To appear, 2011.

[21] D. F. Barrero, M. D. R-Moreno, and D. Camacho. Statistical Estimation of Success Probability in Evolutionary Computation. *Applied Soft Computing*, To appear, 2011.

[22] D. F. Barrero, M. D. R-Moreno, and D. R. López. Information Integration in Searchy: an Ontology and Web Services Approach. *International Journal of Computer Science and Applications (IJCSA)*, 7(2):14–29, 2010.

[23] A. M. Barreto, H. S. Bernardino, and H. J. Barbosa. Probabilistic Performance Profiles for the Experimental Evaluation of Stochastic Algorithms. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation (GECCO 2010)*, pages 751–758, Portland, Oregon, USA, 2010. ACM.

[24] T. Bartz-Beielstein. Tuning Evolutionary Algorithms: Overview and Comprenhensive Introduction. Technical Report 148/03, Universität Dortmund, 2003.

[25] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation: The New Experimentalism*. Natural Computing. Springer, 1 edition, April 2006.

[26] T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss. *Experimental Methods for the Analysis of Optimization Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[27] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. *Experimental Methods for the Analysis of Optimization Algorithms*, chapter The sequential parameter optimization toolbox, pages 337–360. Springer-Verlag New York Inc, 2010.

[28] T. Bartz-Beielstein, C. W. G. Lasarczyk, and M. Preuss. Sequential Parameter Optimization. In *IEEE Congress on Evolutionary Computation (CEC 2005)*, volume 1, pages 773–780. IEEE, 2005.

[29] T. Bartz-Beielstein and S. Markon. Threshold Selection, Hypothesis Tests, and DOE Methods. In *In 2002 Congresss on Evolutionary Computation*, pages 777–782. IEEE Press, 2002.

[30] T. Bartz-Beielstein and M. Preuss. Considerations of Budget Allocation for Sequential Parameter Optimization (SPO). In *Proceedings of the Workshop on Empirical Methods for the Analysis of Algorithms*, pages 35–40, Reykjavik, Iceland, 2006.

[31] T. Bartz-Beielstein and M. Preuss. Tuning and Experimental Analysis in Evolutionary Computation: What we Still Have Wrong. In *Proceedings of the 12th Conference on Genetic and Evolutionary Computation (GECCO 2010)*, pages 2625–2646, Portland, Oregon, USA, 2010. ACM.

[32] J. Baxter. Local Optima Avoidance in Depot Location. *Journal of the Operation Research Society*, 32:815–819, 1981.

[33] D. Beasley, D. R. Bull, and R. R. Martin. An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing*, 15(2):58–69, 1993.

[34] D. Beasley, D. R. Bull, and R. R. Martin. An Overview of Genetic Algorithms: Part 2, Research Topics. *University Computing*, 15(4):170–181, 1993.

[35] H.-G. Beyer and H.-P. Schwefel. Evolution Strategies - A Comprehensive Introduction. *Journal of Natural Computing*, 1:3–52, May 2002.

[36] M. Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2004.

[37] M. Birattari. *Tuning Metaheuristics: a Machine Learning Perspective*, volume 197. Springer-Verlag, 2009.

[38] M. Birattari and M. Dorigo. How to Assess and Report the Performance of a Stochastic Algorithm on a Benchmark Problem: Mean or Best Result on a Number of Runs? *Optimization Letters*, 1(3):309–311, 2007.

[39] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A Racing Algorithm for Configuring Metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[40] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. *Experimental Methods for the Analysis of Optimization Agorithms*, chapter F-Race and iterated F-Race: An overview, pages 311–336. Springer-Verlag, Berlin, Germany, June 2009.

[41] M. Birattari, M. Zlochin, and M. Dorigo. Towards a Theory of Practice in Metaheuristics Design: A Machine Learning Perspective. *Theoretical Informatics and Applications*, 40(2):353–369, 2006.

[42] C. Blum and D. Merkle. *Swarm Intelligence: Introduction and Applications*. Springer-Verlag, 2008.

[43] C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, September 2003.

[44] L. D. Brown, T. T. Cai, and A. Dasgupta. Interval Estimation for a Binomial. *Statistical Science*, 16:101–133, 2001.

[45] L. D. Brown, T. T. Cai, and A. Dasgupta. Confidence Intervals for a Binomial Proportion and Asymptotic Expansions. *Annals of Statistics*, 30(1):160–201, 2002.

[46] J. Brownlee. A Note on Research Methodology and Benchmarking Optimization Algorithms. Technical report, Complex Intelligent Systems Laboratory (CIS), Centre for Information Technology Research (CITR), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, 2007.

[47] D. S. Burke, K. A. D. Jong, J. J. Grefenstette, C. L. Ramsey, and A. S. Wu. Putting More Genetics into Genetic Algorithms. *Evolutionary Computation*, 6:387–410, 1998.

[48] E. K. Burke, S. Gustafson, and G. Kendall. Diversity in Genetic Programming: An Analysis of Measures and Correlation With Fitness. *IEEE Transactions on Evolutionary Computation*, 8:47–62, 2004.

[49] D. Camacho, M. D. R-Moreno, D. F. Barrero, and R. Akerkar. Semantic Wrappers for Semi-Structured Data. *Computing Letters (COLE)*, 4(1-4):21–34, December 2008.

[50] D. Camacho, M. D. R-Moreno, D. F. Barrero, and R. Akerkar. Semantic Wrappers for Semi-structured Data Extraction. *Computing Letters (COLE)*, 4(1):1–14, 2008.

[51] D. Caragea, J. Pathak, J. Bao, A. Silvescu, C. Andorf, D. Dobbs, and V. Honavar. Information Integration and Knowledge Acquisition from Semantically Heterogeneous Biological Data Sources. In *Proceedings of the 16th International Workshop on Database and Expert Systems Applications*, pages 175–190. Springer-Verlag, 2005.

[52] M. Chiarandini, L. Paquete, M. Preuss, and E. Ridge. Experiments on Metaheuristics: Methodological Overview and Open Issues. Technical Report DMF-2007-03-003, The Danish Mathematical Society, Denmark, 2007.

[53] M. Chiarandini and T. Stützle. Experimental Evaluation of Course Timetabling Algorithms. Technical Report AIDA-02-05, Intellectics Group, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany, April 2002.

[54] S. Christensen and F. Oppacher. An Analysis of Koza's Computational Effort Statistic for Genetic Programming. In *Proceedings of the 5th European Conference on Genetic Programming (EuroGP'02)*, pages 182–191, London, UK, 2002. Springer-Verlag.

[55] S. Christensen and F. Oppacher. Solving the Artificial Ant on the Santa Fe Trail Problem in 20,696 Fitness Evaluations. In *Proceedings of the 9th Conference on Genetic and Evolutionary Computation (GECCO 2007)*, pages 1574–1579. ACM, 2007.

[56] D. Chu and J. E. Rowe. Crossover Operators to Control Size Growth in Linear GP and Variable Length GAs. In J. Wang, editor, *IEEE World Congress on Computational Intelligence*, Hong Kong, 1-6 June 2008. IEEE Press.

[57] O. Cicchello and S. C. Kremer. Beyond EDSM. In *Proceedings of the 6th International Colloquium on Grammatical Inference (ICGI 2002)*, pages 37–48, London, UK, 2002. Springer-Verlag.

[58] C. Clopper and S. Pearson. The Use of Confidence or Fiducial Limits Illustrated in the Case of the Binomial. *Biometrika*, 26:404–413, 1934.

[59] P. R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1995.

[60] J. I. Criado. *Las Tecnologías de la Información y la Comunicación en la Modernización de las Administraciones Públicas. Un Análisis de la Configuración de la e-Administración en la Comunidad de Madrid y la Generalitat Valenciana (1995-2005)*. PhD thesis, Universidad Complutense de Madrid, 2009.

[61] C. Cruz, J. González, and D. Pelta. Optimization in Dynamic Environments: A Survey on Problems, Methods and Measures. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 15:1427–1448, 2011.

[62] J. Daida, S. Ross, J. Mcclain, D. Ampy, and M. Holczer. Challenges with Verification, Repeatability, and Meaningful Comparisons in Genetic Programming. In *Proceedings of the Second Annual Conference on Genetic Programming*, pages 64–69. Morgan Kaufmann, 1997.

[63] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favored Races in the Struggle for Life*. John Murray, London, 1859.

[64] S. Das and P. Suganthan. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, February 2011.

[65] R. Dawkins. *The Selfish Gene*. Oxford University Press, USA, 2006.

[66] R. Dawkins. *The Greatest Show on Earth: The Evidence for Evolution*. Transworld Publishers, 2009.

[67] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, Ann Arbor, MI, USA, 1975.

[68] P. Deepti and B. Majumdar. Semantic Web Services in Action - Enterprise Information Integration. In *Proceedings of the 5th international conference on Service-Oriented Computing (ICSOC 2007)*, pages 485–496, Berlin, Heidelberg, 2007. Springer-Verlag.

[69] P. J. Denning. Performance Evaluation: Experimental Computer Science at its Best. In *Proceedings of the ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 1981)*, pages 106–109. ACM, 1981.

[70] P. J. Denning. Is Computer Science Science? *Communications of the ACM*, 48:27–31, April 2005.

[71] J. Derrac, S. García, D. Molina, and F. Herrera. A Practical Tutorial on the Use of Nonparametric Statistical Tests as a Methodology for Comparing Evolutionary and Swarm Intelligence Algorithms. *Swarm and Evolutionary Computation*, 1:3–18, 2011.

[72] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[73] W. Ehm. Binomial Approximation to the Poisson Binomial Distribution. *Statistics & Probability Letters*, 11(1):7–16, January 1991.

[74] A. Eiben and T. Bäck. Empirical Investigation of Multiparent Recombination Operators in Evolution Strategies. *Evolutionary Computation*, 5(3):347–365, 1997.

[75] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 3:124–141, 1999.

[76] A. E. Eiben and M. Jelasity. A Critical Note on Experimental Research Methodology in EC. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002)*, pages 582–587. IEEE, 2002.

[77] A. E. Eiben and C. A. Schippers. On Evolutionary Exploration and Exploitation. *Fundamenta Informaticae*, 35:35–50, August 1998.

[78] A. E. Eiben and S. K. Smit. Parameter Tuning for Configuring and Analyzing Evolutionary Algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[79] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Natural Computing. Springer-Verlag, 2009.

[80] S. Epstein and X. Yun. From Unsolvable to Solvable: An Exploration of Simple Changes. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[81] J. Farmer, N. Packard, and A. Perelson. The Immune System, Adaptation, and Machine Learning. *Physica D: Nonlinear Phenomena*, 22(1-3):187–204, 1986.

[82] T. Feo, M. Resende, and S. Smith. A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. *Operations Research*, pages 860–878, 1994.

[83] T. A. Feo and M. G. C. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, 8(2):67 – 71, 1989.

[84] S. Finck, N. Hansen, R. Ros, and A. Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Presentation of the Noiseless Functions. Technical report 2009/20, Research Center PPE, 2009.

[85] P. J. Fleming and J. J. Wallace. How not to Lie with Statistics: The Correct Way to Summarize Benchmark Results. *Communications of the ACM*, 29:218–221, March 1986.

[86] D. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, volume 1. Wiley-IEEE Press, 2006.

[87] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*, volume 26. John Wiley & Sons, 1966.

[88] C. Fonseca and P. Fleming. On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers. In *Parallel Problem Solving from Nature (PPSN IV)*, pages 584–593. Springer, 1996.

[89] D. Frost, I. Rish, and L. Vila. Summarizing CSP Hardness with Continuous Probability Distributions. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence (AAAI'97/IAAI'97)*, pages 327–333. AAAI Press, 1997.

[90] M. Gagliolo and C. Legrand. *Experimental Methods for the Analysis of Optimization Algorithms*, chapter Algorithm Survival Analysis, pages 161–184. Springer-Verlag, 2010.

[91] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis, Second Edition (Chapman & Hall/CRC Texts in Statistical Science)*. Chapman and Hall/CRC, 2 edition, July 2003.

[92] I. P. Gent, S. A. Grant, E. MacIntyre, P. Prosser, P. Shaw, B. M. Smith, and T. Walsh. How Not To Do It. Research report 97.27, School of Computer Studies, University of Leeds, May 1997.

[93] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research*, 13:533–549, May 1986.

[94] A. V. Goldberg. Selecting Problems for Algorithm Evaluation. In *Proceedings of the 3rd International Workshop on Algorithm Engineering (WAE'99)*, pages 1–11, London, UK, 1999. Springer-Verlag.

[95] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[96] D. E. Goldberg, D. E. Goldberg, K. Deb, K. Deb, H. Kargupta, H. Kargupta, G. Harik, and G. Harik. Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64. Morgan Kaufmann, 1993.

[97] B. W. Goldman and D. R. Tauritz. Self-Configuring Crossover. In *Proceedings of the Conference Companion on Genetic and Evolutionary Computation (GECCO 2011)*, pages 575–582. ACM, 2011.

[98] A. González, D. F. Barrero, M. D. R-Moreno, and D. Camacho. A Case Study on Grammatical-Based Representation for Regular Expression Evolution. In *Proceedings of 8th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2010)*, volume 2, pages 379–386, Salamanca, Spain, 26-28 April 2010. Springer-Verlag.

[99] M. Graff and R. Poli. Practical Model of Genetic Programming's Performance on Rational Symbolic Regression Problems. In *Proceedings of the 11th European conference on Genetic programming (EuroGP 2008)*, pages 122–133, Berlin, Heidelberg, 2008. Springer-Verlag.

[100] T. Grubber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[101] S. Gustafson, A. Ekárt, E. Burke, and G. Kendall. Problem Difficulty and Code Growth in Genetic Programming. *Genetic Programming and Evolvable Machines*, 5(3):271–290, 2004.

[102] L. Haas. Beauty and the Beast: The Theory and Practice of Information Integration. *Proceedings of the 11th International Conference on Database Theory (ICDT 2007)*, 4353:28–43, Jan 2007.

[103] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-Parameterblack-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Technical report RR-6829, INRIA, 2009.

[104] P. Hansen and N. Mladenovic. Variable Neighborhood Search: Principles and Applications. *European Journal Of Operational Research*, 130(3):449–467, 2001.

[105] I. Harvey. The SAGA Cross: The Mechanics of Recombination for Species with Variablelength Genotypes. In *Parallel Problem*, pages 269–278. North-Holland, 1992.

[106] F. Herrera, M. Lozano, and D. Molina. Test Suite for the Special Issue of Soft Computing on Scalability of Evolutionary Algorithms and Other Metaheuristics for Large Scale Continuous Optimization Problems. 2010.

[107] K. E. Hillstrom. A Simulation Test Approach to the Evaluation of Nonlinear Optimization Algorithms. *ACM Transactions on Mathematical Software*, 3:305–315, December 1977.

[108] R. Hinterding, Z. Michalewicz, and A. Eiben. Adaptation in Evolutionary Computation: A Survey. In *IEEE International Conference on Evolutionary Computation (CEC 1997)*, pages 65–69. IEEE, 1997.

[109] J. L. Hodges and L. Le Cam. The Poisson Approximation to the Poisson Binomial Distribution. *The Annals of Mathematical Statistics*, 31(3):737–740, September 1960.

[110] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 1992.

[111] J. Hooker. Needed: An Empirical Science of Algorithms. *Operations Research*, 42:201–212, 1994.

[112] J. Hooker. Testing Heuristics: We Have it All Wrong. *Journal of Heuristics*, 1:33–42, 1995.

[113] H. Hoos and T. Stützle. Characterizing the Run-Time Behavior of Stochastic Local Search. In *Proceedings AAAI99*. Citeseer, 1998.

[114] H. Hoos and T. Stützle. Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT. *Artificial Intelligence*, 112(1-2):213–232, 1999.

[115] H. Hoos and T. Stützle. Local Search Algorithms for SAT: An Empirical Evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

[116] H. H. Hoos and T. Stützle. Evaluating Las Vegas Algorithms – Pitfalls and Remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 238–245. Morgan Kaufmann Publishers, 1998.

[117] B. Hutt and K. Warwick. Synapsing Variable-Length Crossover: Meaningful Crossover for Variable-Length Genomes. *IEEE Transactions on Evolutionary Computation*, 11(1):118–131, 2007.

[118] C. Igel and K. Chellapilla. Investigating the Influence of Depth and Degree of Genotypic Change on Fitness in Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, pages 1061–1068, Orlando, Florida, USA, July 13-17 1999.

[119] T. Jansen and C. Zarges. Comparing Different Aging Operators. In *Proceedings of the 8th International Conference on Artificial Immune Systems*, ICARIS '09, pages 95–108, Berlin, Heidelberg, 2009. Springer-Verlag.

[120] Y. Jin and J. Branke. Evolutionary Optimization in Uncertain Environments-a Survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[121] C. G. Johnson. Genetic Programming Crossover: Does It Cross over? In *Proceedings of the 12th European Conference on Genetic Programming (EuroGP 2009)*, pages 97–108. Springer-Verlag, 2009.

[122] D. Johnson. A Theoretician's Guide to the Experimental Analysis of Algorithms. *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, 59:215–250, 2002.

[123] I. Kant. *The Critique of Pure Reason*. Cambridge University Press, 1999.

[124] D. Karaboga and B. Basturk. A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm. *Journal of Global Optimization*, 39(3):459–471, 2007.

[125] A. Kaufmann, D. Grounchko, and R. Cruon. *Mathematical Models for the Study of the Reliability of Systems*, volume 124 of *Mathematics in Science and Engineering*. Academic Press, Inc., 1977.

[126] A. Kaveh and S. Talatahari. A Novel Heuristic Optimization Method: Charged System Search. *Acta Mechanica*, 213(3-4):267–289, 2010.

[127] M. Keijzer, V. Babovic, C. Ryan, M. O'Neill, and M. Cattolico. Adaptive Logic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 42–49, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[128] J. Kennedy and R. Eberhart. Particle Swarm Optimization. In *IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[129] L. Kerschberg, M. Chowdhury, A. Damiano, H. Jeong, S. Mitchell, J. Si, and S. Smith. Knowledge Sifter: Ontology-Driven Search over Heterogeneous Databases. In *16th International Conference on Scientific and Statistical Database Management (SSDBM 2004)*, pages 431–432, Santorini Island, Greece, June 2004. IEEE Computer Society.

[130] D. Kinzett, M. Johnston, and M. Zhang. How Online Simplification Affects Building Blocks in Genetic Programming. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation (GECCO 2009)*, pages 979–986, New York, NY, USA, 2009. ACM.

[131] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671, 1983.

[132] C. A. Knoblock and J.-L. Ambite. Agents for Information Gathering. In *Software Agents*, pages 347–374. AAAI Press / The MIT Press, 1997.

[133] J. Knowles, D. Corne, and K. Deb. *Multiobjective Problem Solving from Nature: from Concepts to Applications*. Springer-Verlag, 2008.

[134] J. Koza. *Genetic Programming II: automatic discovery of reusable programs*. MIT Press, 1994.

[135] J. Koza. Human-Competitive Results Produced by Genetic Programming. *Genetic Programming and Evolvable Machines*, 11(3):251–284, September 2010.

[136] J. R. Koza. *Genetic Programming: On the programming of Computers by Means of Natural Selection*. MIT Press, Cambrige, MA, 1992.

[137] J. R. Koza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[138] J. R. Koza, D. Andre, F. H. Bennett, and M. A. Keane. *Genetic Programming III: Darwinian Invention & Problem Solving*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[139] K. Krishnanand and D. Ghose. Detection of Multiple Source Locations Using a Glowworm Metaphor with Applications to Collective Robotics. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2005)*, pages 84–91. IEEE, 2005.

[140] G. Kronberger, S. Winkler, M. Affenzeller, and S. Wagner. On Crossover Success Rate in Genetic Programming with Offspring Selection. In *Proceedings of the 12th European Conference on Genetic Programming (EuroGP 2009)*, pages 232–243, Berlin, Heidelberg, 2009. Springer-Verlag.

[141] K. J. Lang. Evidence Driven State Merging with Search. Technical report, NEC Research Institute, 1998.

[142] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm. In *Proceedings of the 4th International Colloquium on Grammatical Inference (ICGI 1998)*, pages 1–12, London, UK, 1998. Springer-Verlag.

[143] W. B. Langdon and R. Poli. Why Ants are Hard. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 193–201, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[144] W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. *The Evolution of Size and Shape*, pages 163–190. MIT Press, Cambridge, MA, USA, 1999.

[145] P.-S. Laplace. *Théorie Analytique des probabilités*. Mme. Ve Courcier, Paris, France, 1812.

[146] P. Larranaga and J. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, volume 2. Springer-Verlag, 2002.

[147] C.-Y. Lee and E. K. Antonsson. Variable Length Genomes for Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000)*, Las vegas, Nevada, USA, 2000.

[148] L. Leemis and K. S. Trivedi. A Comparison of Approximate Interval Estimators for the Bernoulli Parameter. Technical report, 1993.

[149] J. Liang, T. Runarsson, E. Mezura-Montes, M. Clerc, P. Suganthan, C. Coello, and K. Deb. Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-parameter Optimization. Technical report, Nangyang Technological University, Singapore, Tech. Rep., 2006.

[150] E. Limpert, W. A. Stahel, and M. Abbt. Log-normal Distributions across the Sciences: Keys and Clues. *BioScience*, 51(5):341–352, May 2001.

[151] F. G. Lobo and C. F. Lima. A Review of Adaptive Population Sizing Schemes in Genetic Algorithms. In *Proceedings of the 2005 workshops on Genetic and evolutionary computation (GECCO 2005)*, pages 228–234. ACM, 2005.

[152] J. Lozano. *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms*, volume 192. Springer-Verlag New York Inc, 2006.

[153] S. M. Lucas and T. J. Reynolds. Learning DFA: Evolution Versus Evidence Driven State Merging. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2003)*, pages 351–358, Newport Beach, California, USA, 2003.

[154] S. Luke. A Java-based Evolutionary Computation Research System (ECJ Libraries) home page. http://cs.gmu.edu/˜eclab/projects/ecj/.

[155] S. Luke. Code Growth Is Not Caused by Introns. In *In Whitley, D. (Ed.), Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference. Las Vegas*, pages 228–235. Morgan Kaufmann, 2000.

[156] S. Luke. When Short Runs Beat Long Runs. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 74–80. Morgan Kaufmann, 2001.

[157] S. Luke. Modification Point Depth and Genome Growth in Genetic Programming. *Evolutionary Computation*, 11(1):67–106, 2003.

[158] S. Luke. *Essentials of Metaheuristics*. Lulu Enterprises, UK Ltd, 2009.

[159] S. Luke and L. Panait. Is the Perfect the Enemy of the Good? In *In Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 820–828, New York, New York, USA, 2002. Morgan Kaufmann.

[160] S. Luke and L. Panait. A Comparison of Bloat Control Methods for Genetic Programming. *Evolutionary Computation*, 14:309–344, September 2006.

[161] S. Luke and L. Spector. A Revised Comparison of Crossover and Mutation in Genetic Programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 240–248. Morgan Kaufmann, 1998.

[162] R. Mallipeddi and P. Suganthan. Problem Definitions and Evaluation Criteria for the CEC 2010 Competition on Constrained Real-Parameter Optimization. Technical report, Nanyang Technological University, Singapore, Tech. Rep., 2009.

[163] O. Maron and A. Moore. The Racing Algorithm: Model Selection for Lazy Learners. *Artificial Intelligence Review*, 11(1):193–225, 1997.

[164] R. W. Matthews and J. R. Matthews. *Insect Behavior*. Springer-Verlag, 2009.

[165] C. McGeoch. Toward an Experimental Method for Algorithm Simulation. *INFORMS Journal on Computing*, 8(1):1–15, 1996.

[166] R. I. Mckay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill. Grammar-Based Genetic Programming: A Survey. *Genetic Programming and Evolvable Machines*, 11:365–396, September 2010.

[167] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[168] M. Michalowski, J. Ambite, S. Thakkar, R. Tuchinda, C. Knoblock, and S. Minton. Retrieving and Semantically Integrating Heterogeneous Data from the Web. *IEEE Intelligent Systems*, 19(3), 2004.

[169] J. Miller. An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Programming Approach. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, volume 2, pages 1135–1142, 1999.

[170] J. Miller and P. Thomson. Cartesian Genetic Programming. In *Genetic Programming*, volume 1802 of *Lecture Notes in Computer Science*, pages 121–132. Springer-Verlag / Heidelberg, 2000.

[171] C. R. M'Lan, J. Lawrence, and D. B. Wolfson. Bayesian sample size determination for binomial proportions. *Bayesian Analysis*, 3(2):269–296, 2008.

[172] D. Montana. Strongly Typed Genetic Programming. *Evolutionary computation*, 3(2):199–230, 1995.

[173] D. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons Inc, 1984.

[174] D. C. Montgomery and G. C. Runger. *Applied Statistics and Probability for Engineers, 4th Edition*. John Wiley & Sons, 4th edition, May 2010.

[175] B. Moret. Towards a Discipline of Experimental Algorithmics. In *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, volume 59, pages 197–214, 2002.

[176] R. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. Springer-Verlag, 2004.

[177] R. Myers and E. R. Hancock. Empirical Modelling of Genetic Algorithms. *Evolutionary Computation*, 9(4):461–493, 2001.

[178] V. Nannen and A. Eiben. Efficient Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. In *IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 103–110. IEEE, 2007.

[179] R. G. Newcombe. Two-Sided Confidence Intervals for the Single Proportion: Comparison of Seven Methods. *Statistics in Medicine*, 17(8):857–872, 1998.

[180] J. Niehaus and W. Banzhaf. More on Computational Effort Statistics for Genetic Programming. In *Proceedings of the European Conference on Genetic Programming (EuroGP 2003)*, volume 2610 of *Lecture Notes on Computer Science*, pages 164–172, Essex, UK, 14-16 Apr. 2003. Springer-Verlag.

[181] M. H. Nodine, J. Fowler, T. Ksiezyk, T. Perry, M. Taylor, and A. Unruh. Active Information Gathering in InfoSleuth. *International Journal of Cooperative Information Systems*, 9(1-2):3–28, 2000.

[182] P. Norvig. Warning Signs in Experimental Design and Interpretation. http://norvig.com/experiment-design.html.

[183] N. F. Noy. Semantic Integration: A Survey of Ontology-Based Approaches. *SIGMOD Rec.*, 33(4):65–70, December 2004.

[184] M. O'Neill and C. Ryan. Grammatical Evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, August 2001.

[185] M. ONeill, L. Vanneschi, S. Gustafson, and W. Banzhaf. Open Issues in Genetic Programming. *Genetic Programming and Evolvable Machines*, 11(3-4):339–363, 2010.

[186] K. Passino. Biomimicry of bacterial foraging for distributed optimization and control. *Control Systems Magazine, IEEE*, 22(3):52–67, 2002.

[187] N. Paterson and M. Livesey. Performance Comparison in Genetic Programming. In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference (GECCO 2000)*, pages 253–260, Las Vegas, Nevada, USA, July 2000.

[188] P. Pellegrini and M. Birattari. Implementation Effort and Performance: A Comparison of Custom and Out-of-the-Box Metaheuristics on the Vehicle Routing Problem With Stochastic Demand. In *Proceedings of the 2007 International Conference on Engineering Stochastic Local Search Algorithms: Designing, Implementing and Analyzing Effective Heuristics*, SLS'07, pages 31–45, Berlin, Heidelberg, 2007. Springer-Verlag.

[189] W. W. Piegorsch. Sample Sizes for Improved Binomial Confidence Intervals. *Computational Statistics & Data Analysis*, 46(2):309–316, June 2004.

[190] A. M. Pires and C. a. Amado. Interval Estimators for a Binomial Proportion: Comparison of Twenty Methods. *Statistical Journal*, 6(2):165–197, Jun 2008.

[191] Plato. *The Republic*. NTC/Contemporary Publishing Company, Sept. 1997.

[192] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, Mar. 2008.

[193] R. Poli, L. Vanneschi, W. Langdon, and N. McPhee. Theoretical Results in Genetic Programming: The Next Ten years? *Genetic Programming and Evolvable Machines*, 11(3):285–320–320, September 2010.

[194] M. R-Moreno, D. Camacho, D. Barrero, and B. Castaño. Human Drivers Knowledge Integration in a Logistics Decision Support Tool. In *Intelligent Distributed Computing V*, volume 382 of *Studies in Computational Intelligence*, pages 227–236. Springer-Verlag / Heidelberg, 2011.

[195] M. D. R-Moreno, D. Camacho, D. F. Barrero, and M. Gutiérrez. A Decision Support System for Logistics Operations. In *Soft Computing Models in Industrial and Environmental Applications, 5th International Workshop (SOCO 2010)*, volume 73 of *Advances in Soft Computing*, pages 103–110, Guimarães, Portugal, 2010. Springer-Verlag.

[196] M. D. R-Moreno, B. Castaño, M. Carbajo, Á. Moreno, D. F. Barrero, and P. Muñoz. Multi-Agent Intelligent Planning Architecture for People Location and Orientation Using RFID. *Cybernetics and Systems*, 42(1):16–32, Jan 2011.

[197] P. Rabanal, I. Rodríguez, and F. Rubio. Using River Formation Dynamics to Design Heuristic Algorithms. *Unconventional Computation*, pages 163–177, 2007.

[198] E. Rahme, L. Joseph, and T. W. Gyorkos. Bayesian Sample Size Determination for Estimating Binomial Parameters from Data Subject to Misclassification. *Journal Of The Royal Statistical Society Series C*, 49(1):119–128, 2000.

[199] M. Ramilo Araujo. *Políticas públicas, instituciones y actores para la promoción de la sociedad de la informacion y/o del conocimiento. Un análisis comparado de Catalunya y Euskadi*. PhD thesis, Universidad del País Vasco, 2009.

[200] W. Rand and R. Riolo. Measurements for understanding the behavior of the genetic algorithm in dynamic environments: a case study using the Shaky Ladder Hyperplane-Defined Functions. In *Proceedings of the Workshops on Genetic and Evolutionary Computation (GECCO 2005)*, pages 32–38. ACM, 2005.

[201] R. L. Rardin and R. Uzsoy. Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial. *Journal of Heuristics*, 7:261–304, May 2001.

[202] I. Rechenberg. *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Number 15 in Problemata. Frommann-Holzboog, Stuttgart-Bad Cannstatt, 1973.

[203] C. R. Reeves and C. Wright. Genetic Algorithms and The Design of Experiments. In *Proceedings of the IMA Fall Workshop on Evolutionary Algorithms*, 1996.

[204] G. Reinelt. TSPLIB-A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[205] C. C. Ribeiro, I. Rosseti, and R. Vallejos. On the Use of Run Time Distributions to Evaluate and Compare Stochastic Local Search Algorithms. In *Proceedings of the Second International Workshop on Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics (SLS'09)*, pages 16–30. Springer-Verlag, 2009.

[206] E. Ridge. *Design of Experimetns for the Tuning of Optimization Algorithms*. PhD thesis, The University of York. Department of Computer Science, October 2007.

[207] T. D. Ross. Accurate Confidence Intervals for Binomial Proportion and Poisson Rate Estimation. *Computers in Biology and Medicine*, 33(6):509–531, 2003.

[208] F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Springer-Verlag, Heidelberg, New York, 2nd edition edition, 2006.

[209] B. Russell. *A History of Western Philosophy*. Touchstone, 1945.

[210] S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice hall, 2010.

[211] A. D. Sarma, X. Dong, and A. Halevy. Bootstrapping Pay-as-you-go Data Integration Systems. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD 2008)*, pages 861–874, New York, NY, USA, 2008. ACM.

[212] H. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., 1993.

[213] H. Shah-Hosseini. The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm. *International Journal of Bio-Inspired Computation*, 1(1):71–79, 2009.

[214] R. Sharma. Bayes Approach to Interval Estimation of a Binomial Parameter. *Annals of the Institute of Statistical Mathematics*, 27(1):259–267, 1975.

[215] S. Silva and E. Costa. Dynamic Limits for Bloat Control in Genetic Programming and a Review of Past and Current Bloat Theories. *Genetic Programming and Evolvable Machines*, 10(2):141–179, 2009.

[216] S. Smit and A. Eiben. *Experimental Methods for the Analysis of Optimization Algorithms*, chapter Using Entropy for Parameter Analysis of Evolutionary Algoirthms, pages 287–308. Springer-Verlag New York Inc, 2010.

[217] S. Smit and A. Eiben. Parameter Tuning of Evolutionary Algorithms: Generalist vs. Specialist. In *Applications of Evolutionary Computation*, volume 6024 of *Lecture Notes in Computer Science*, pages 542–551, Berlin, Heidelberg, 2010. Springer Berlin / Heidelberg.

[218] M. F. Smith. Sampling Considerations In Evaluating Cooperative Extension Programs. In *Florida Cooperative Extension Service Bulletin PE-1*. Institute of Food and Agricultural Sciences. University of Florida., 1983.

[219] W. M. Spears. Repository of Test Problem Generators. http://www.cs.uwyo.edu/ wspears/generators.html.

[220] W. M. Spears. Crossover or Mutation. *Foundations of Genetic Algorithms 2*, 2:221–237, 1993.

[221] M. Srinivas and L. Patnaik. Genetic Algorithms: A Survey. *Computer*, 27(6):17–26, 1994.

[222] P. F. Stadler. Fitness Landscapes. *Biological Evolution and Statistical Physics*, 585:183–204, 2002.

[223] R. Storn and K. Price. Differential Evolution-a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. *International Computer Science Institute-Publications-TR*, 1995.

[224] T. Stützle and H. H. Hoos. Analyzing the Run-Time Behaviour of Iterated Local Search for the TSP. In *III Metaheuristics International Conference*. Kluwer Academic Publishers, 1999.

[225] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y. Chen, A. Auger, and S. Tiwari. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. *KanGAL Report*, 2005.

[226] E.-G. Talbi. A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics*, 8:541–564, September 2002.

[227] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise. Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization. Technical report, University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL): Héféi, Ānhuī, China, 2009.

[228] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang. Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, Nanyang Technology University, Singapore, China, 2007.

[229] M. Tomita. Dynamic Construction of Finite Automata from Examples Using Hill Climbing. In *Proceedings of the Fourth Annual Cognitive Science Conference*, pages 105–108, 1982.

[230] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley and Sons Ltd., Chichester, UK, 2002.

[231] A. Turing. Computing Machinery and Intelligence. *Mind*, 59(236):433–460, 1950.

[232] M. Uschold and M. Gruninger. Ontologies and Semantics for Seamless Connectivity. *SIGMOD Rec.*, 33(4):58–64, December 2004.

[233] L. Vanneschi. *Theory and Practice for Efficient Genetic Programming*. PhD thesis, University de Lausanne, 2004.

[234] R. Vdovjak and G.-J. Houben. RDF-Based Architecture for Semantic Integration of Heterogeneous Information Sources. In *Workshop on Information Integration on the Web*, pages 51–57, 2001.

[235] S. E. Vollset. Confidence Intervals for a Binomial Proportion. *Statistics in Medicine*, 12(9):809–827, 1993.

[236] S. Voß. Meta-Heuristics: The State of the Art. In *Proceedings of the Workshop on Local Search for Planning and Scheduling-Revised Papers (ECAII 2000)*, pages 1–23, London, UK, 2001. Springer-Verlag.

[237] C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. PhD thesis, Department of Computer Science, University of Essex, 1997.

[238] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-Based Integration of Information — a Survey of Existing Approaches. In *Workshop on Ontologies and Information Sharing (IJCAI 2001)*, pages 108–117, Seattle, Washington, USA, 2001.

[239] A. Wald. Test of Statistical Hypotheses Concerning Several Parameters when the Number of Observations is Large. *Transactions of the American Mathematical Society*, 54(3):426–482, Nov 1943.

[240] M. Walker, H. Edwards, and C. Messom. The Reliability of Confidence Intervals for Computational Effort Comparisons. In *Proceedings of the 9th Conference on Genetic and Evolutionary Computation (GECCO 2007)*, pages 1716–1723. ACM, 2007.

[241] M. Walker, H. Edwards, and C. H. Messom. Confidence Intervals for Computational Effort Comparisons. In *E uroGP*, pages 23–32, 2007.

[242] M. Walker, H. Edwards, and C. H. Messom. Success Effort and Other Statistics for Performance Comparisons in Genetic Programming. In *IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 4631–4638, Singapore, 2007. IEEE.

[243] K. Weicker. Performance Measures for Dynamic Environments. In *Parallel Problem Solving from Nature PPSN VII*, pages 64–73. Springer-Verlag, 2002.

[244] D. R. White and S. Poulding. A Rigorous Evaluation of Crossover and Mutation in Genetic Programming. In *Proceedings of the 12th European Conference on Genetic Programming (EuroGP 2009)*, pages 220–231. Springer-Verlag, 2009.

[245] D. Whitley. An Overview of Evolutionary Algorithms: Practical Issues and Common Pitfalls. *Information and Software Technology*, 43(14):817–831, Dec. 2001.

[246] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Evaluating Evolutionary Algorithms. *Artificial Intelligence*, 85:245–276, 1996.

[247] E. B. Wilson. Probable Inference, the Law of Succession, and Statistical Inference. *Journal of the American Statistical Association*, (22):309–316, 1927.

[248] M. Wineberg and S. Christensen. Statistical Analysis for Evolutionary Computation: Advanced Techniques. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation (GECCO 2010)*, pages 2661–2682. ACM, 2010.

[249] M. Wineberg and S. Christensen. Statistical Analysis for Evolutionary Computation: Introduction. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation (GECCO 2010)*, pages 2413–2440. ACM, 2010.

[250] D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.

[251] A. S. Wu and I. Garibay. The Proportional Genetic Algorithm: Gene Expression in a Genetic Algorithm. *Genetic Programming and Evolvable Machines*, 3(2):157–192, 2002.

[252] X. Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2010.

[253] J. Yuan, A. Bahrami, C. Wang, M. Murray, and A. Hunt. A Semantic Information Integration Tool Suite. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB 2006)*, pages 1171–1174. VLDB Endowment, 2006.

[254] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang. Multiobjective Evolutionary Algorithms: A Survey of the State of the Art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.

[255] F. Zhu, M. Turner, I. A. Kotsiopoulos, K. H. Bennett, M. Russell, D. Budgen, P. Brereton, M. R. John Keane and, and J. Xu. Dynamic Data Integration Using Web Services. In *IEEE International Conference on Web Services (ICWS'04)*, pages 262–269, San Diego, California, USA, June 2004. IEEE Computer Society.

[256] G. K. Zipf. *The Psychobiology of Language*. Houghton-Mifflin, New York, NY, USA, 1935.

[257] E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8:173–195, June 2000.

[258] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.