Pontifícia Universidade Católica
do Rio de Janeiro

Luisa Zambelli Artmann Rangel Vilela

Strategies for Parameter Control in the Biased Random-Key Genetic Algorithm

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-graduação em Engenharia de Produção of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia de Produção.

Advisor : Prof. Luciana de Souza Pessôa
Co-advisor: Dr. Carlos Eduardo de Andrade

Rio de Janeiro
October 2022

Pontifícia Universidade Católica
do Rio de Janeiro

Luisa Zambelli Artmann Rangel Vilela

Strategies for Parameter Control in the Biased Random-Key Genetic Algorithm

Dissertation presented to the Programa de Pós-graduação em Engenharia de Produção of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia de Produção. Approved by the Examination Committee.

Prof. Luciana de Souza Pessôa
Advisor
Departamento de Engenharia Industrial – PUC-Rio

Dr. Carlos Eduardo de Andrade
Co-advisor
AT&T Labs Research – AT&T

Prof. Fábio Luiz Usberti
UNICAMP

Dr. Mauricio Guilherme de Carvalho Resende
Amazon.com

Rio de Janeiro, October 3rd, 2022

Luisa Zambelli Artmann Rangel Vilela

Graduated in Industrial Engineering at CEFET-RJ. During graduation, she received a Scientific Initiation Scholarship (CNPq) and worked in the implementation of algorithms to support research in Spectral Graph Theory. She spent a semester as a student exchange at the Faculty of Engineering of the University of Porto, where studied subjects such as Artificial Intelligence, Databases, and Systems Modeling and Simulation.

Bibliographic data

Vilela, Luisa Zambelli Artmann Rangel

Strategies for parameter control in the biased random-key genetic algorithm / Luisa Zambelli Artmann Rangel Vilela; advisor: Luciana de Souza Pessôa; co-advisor: Carlos Eduardo de Andrade. – 2022.
115 f. : il. color. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Industrial, 2022.
Inclui bibliografia

1. Engenharia de Produção – Teses. 2. Algoritmo genético de chaves aleatórias enviesadas. 3. Parametrização online. 4. Otimização combinatória. I. Pessôa, Luciana de Souza. II. Andrade, Carlos Eduardo de. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Industrial. IV. Título.

CDD: 658.5

Acknowledgments

I am grateful to God. For the unconditional love of Christ and for giving purpose to my life, allowing me to cooperate with Him in His works.

To my loving husband, João, who inspires me to be better every day. For his amazing support, and for being there for me when I needed the most.

To my father, Luiz. For insisting on the foundations that helped drive me into who I am today. To my mother, Paula. For the unconditional support and encouragement throughout every phase of my life.

To my advisors, Luciana and Carlos. For their guidance along the way, which made this work possible.

To my brother, Pedro, and in-laws, Wilson and Rossana. For their always-present support and care. To my family of faith. For their love and support.

And to Sara, my daughter, who I can't wait to meet.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Vilela, Luisa Zambelli Artmann Rangel; Pessôa, Luciana de Souza (Advisor); Andrade, Carlos Eduardo (Co-Advisor). **Strategies for Parameter Control in the Biased Random-Key Genetic Algorithm**. Rio de Janeiro, 2022. 115p. Dissertação de Mestrado – Departamento de Engenharia Industrial, Pontifícia Universidade Católica do Rio de Janeiro.

The Biased Random-Key Genetic Algorithm (BRKGA) is a population-based metaheuristic applied to obtain optimal or near-optimal solutions to combinatorial problems. To ensure the good performance of this algorithm (and other metaheuristics in general), defining parameter settings is a crucial step. Parameter values have a great influence on determining whether a good solution will be found by the algorithm and whether the search process will be efficient. One way of tackling the parameter setting problem is through the parameter control (or online tuning) approach. Parameter control allows the algorithm to adapt parameter values according to different stages of the search process and to accumulate information on the fitness landscape during the search to use this information in later stages. It also releases the user from the task of defining parameter settings, implicitly solving the tuning problem. In this work, we evaluate two strategies to implement parameter control in BRKGA. Our first approach was adopting random parameter values for each of BRKGA's generations. The second approach was to introduce the principles adopted by Iterated Race, a state-of-the-art tuning method, to BRKGA. Both strategies were evaluated in three classical optimization problems (Flowshop Permutation Problem, Set Covering Problem, and the Traveling Salesman Problem) and led to competitive results when compared to the tuned algorithm.

Keywords

Biased Random-Key Genetic Algorithm; Parameter Control; Combinatorial Optimization.

Resumo

Vilela, Luisa Zambelli Artmann Rangel; Pessôa, Luciana de Souza; Andrade, Carlos Eduardo. **Estratégias para o Controle de Parâmetros no Algoritmo Genético com Chaves Aleatórias Enviesadas**. Rio de Janeiro, 2022. 115p. Dissertação de Mestrado – Departamento de Engenharia Industrial, Pontifícia Universidade Católica do Rio de Janeiro.

O Algoritmo Genético de Chaves Aleatórias Enviesadas (BRKGA) é uma metaheurística populacional utilizada na obtenção de soluções ótimas ou quase ótimas para problemas de otimização combinatória. A parametrização do algoritmo é crucial para garantir seu bom desempenho. Os valores dos parâmetros têm uma grande influência em determinar se uma boa solução será encontrada pelo algoritmo e se o processo de busca será eficiente. Uma maneira de resolver esse problema de configuração de parâmetros é por meio da abordagem de parametrização online (ou controle de parâmetros). A parametrização online permite que o algoritmo adapte os valores dos parâmetros de acordo com os diferentes estágios do processo de busca e acumule informações sobre o espaço de soluções nesse processo para usar as informações obtidas em estágios posteriores. Ele também libera o usuário da tarefa de definir as configurações dos parâmetros, resolvendo implicitamente o problema de configuração. Neste trabalho, avaliamos duas estratégias para implementar o controle de parâmetros no BRKGA. Nossa primeira abordagem foi adotar valores de parâmetros aleatórios para cada geração do BRKGA. A segunda abordagem foi incorporar os princípios adotados pelo irace, um método de parametrização do estado da arte, ao BRKGA. Ambas as estratégias foram avaliadas em três problemas clássicos de otimização (Problema de Permutação Flowshop, Problema de Cobertura de Conjuntos e Problema do Caixeiro Viajante) e levaram a resultados competitivos quando comparados ao algoritmo tunado.

Palavras-chave

Algoritmo genético de chaves aleatórias enviesadas; Parametrização online; Otimização combinatória.

Table of contents

1 Introduction **14**

2 Related Literature **17**
2.1 Biased Random-Key Genetic Algorithm 17
2.1.1 Additional Features 18
2.1.2 Applications 20
2.2 Parameter Settings 23
2.2.1 Parameter Tuning 24
2.2.1.1 Iterated Race 26
2.2.2 Parameter Control 30
2.3 Concluding Remarks 34

3 Proposed Methods **37**
3.1 Random Parameter Values 37
3.2 BRKGA-Race 40
3.2.1 Algorithm and race setup 42
3.2.2 The first race 44
3.2.3 Population selection and individuals' migration 44
3.2.4 Following races 47

4 Experiments and Discussion **49**
4.1 Benchmark Problems 49
4.1.1 Flowshop Scheduling Problem 50
4.1.2 Traveling Salesman Problem 51
4.1.3 Set Covering Problem 52
4.2 Computational Environment 53
4.3 Setting the Use Case 54
4.4 Random Parameter Values 55
4.5 BRKGA-Race 65
4.6 BRKGA-Race: Example Case 65
4.7 BRKGA-Race: Results 68

5 Conclusions **80**

6 References **83**

A Instance's Dimensions **91**
A.1 Flowshop Scheduling Problem 91
A.2 Set Covering Problem 93

B Complete Results for the FSP **95**

C Complete Results for the TSP **104**

D Complete Results for the Set Covering Problem **111**

List of figures

Figure 2.1 Transition between generations on BRKGA. Adapted from [1]. 18
Figure 2.2 Main steps performed by irace. Adapted from [2]. 26
Figure 2.3 A race illustration. The rows represent the instances, and the columns represent the configurations. Each node is an evaluation of one configuration on one instance. On the right, "X" indicates that no statistical test was performed, "–" shows that the test eliminated at least one configuration, and "=" indicates that the test did not discard any configuration. Adapted from López-Ibáñez et al. [2]. 27

Figure 3.1 Flowchart of the BRKGA-Race method. 42
Figure 3.2 Comparing a pair of improvement series and evaluating the dominance of A over B. 45
Figure 3.3 Illustrations of Selection and Migration procedures. On the left, we can see the selection of 2 populations (A and B) in a set of 6 populations that were evaluated after a race. On the right, we see an illustration of the best chromosomes from the non-surviving populations (C, D, E, and F) being migrated into the surviving ones. 47

Figure 4.1 Decoding of a chromosome into a feasible solution of the FSP. Adapted from [3]. 51
Figure 4.2 Decoding of a chromosome into a feasible solution of the TSP. 52
Figure 4.3 Decoding of a chromosome into a preliminary solution of the SCP. 53
Figure 4.4 Distribution of relative percentage deviations from the best-known solution for the FSP instances. 57
Figure 4.5 Distribution of relative percentage deviations from the best-known solution for the TSP instances. 58
Figure 4.6 Distribution of relative percentage deviations from the best-known solution for the SCP instances. 59
Figure 4.7 Distribution of relative percentage deviations from the best-known solution for the FSP instances (with Local Search). 61
Figure 4.8 Distribution of relative percentage deviations from the best-known solution for the TSP instances (with Local Search). 62
Figure 4.9 Distribution of relative percentage deviations from the best-known solution for the SCP instances (with Local Search). 64
Figure 4.10 Evolution of the best cost throughout the evaluations of different configurations on the FSP (instance TA10). 71
Figure 4.11 Boxplot comparing the deviations from the best known solutions of the three evaluated methods with local search on the FSP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race. 72
Figure 4.12 Boxplot comparing the deviations from the best known solutions of the three evaluated methods without local search on the FSP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race. 72
Figure 4.13 Evolution of the best cost throughout the evaluations of different configurations on the TSP (instance brazil58). 75

Figure 4.14 Boxplot comparing the deviations from the best known solutions of the three evaluated methods with local search on the TSP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race. 76
Figure 4.15 Boxplot comparing the deviations from the best known solutions of the three evaluated methods without local search on the TSP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race. 76
Figure 4.16 Evolution of the best cost throughout the evaluations of different configurations on the SCP (instance scp55). 78
Figure 4.17 Boxplot comparing the deviations from the best known solutions of the three evaluated methods with local search on the SCP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race. 78
Figure 4.18 Boxplot comparing the deviations from the best known solutions of the three evaluated methods without local search on the SCP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race. 79

List of tables

Table 2.1 Parameter tuning methods adopted for BRKGA's configuration. 25
Table 2.2 Recommended parameter ranges by Gonçalves and Resende [1]. 25
Table 2.3 Summary of BRKGA applications and parameter settings methodologies. 36

Table 3.1 Two series of solution values throughout the algorithm's execution with different lengths. 45
Table 3.2 Matching size of Series A with Series B. 46

Table 4.1 Elapsed time (in hours and days) to tune each problem. A decoder of type "NLS" indicates the pure decoder, without local search. While "LS" indicates the version with local search included. (*) Indicates that it was not possible to terminate Irace, after several days of execution. 54
Table 4.2 Aggregated results of BRKGA-Random NLS compared to BRKGA-Tuned NLS for the three studied problems. 56
Table 4.3 Results of BRKGA-Random NLS compared to BRKGA-Tuned NLS for the Flowshop Scheduling Problem. 57
Table 4.4 Results of BRKGA-Random NLS compared to BRKGA-Tuned NLS for the Traveling Salesman Problem. 58
Table 4.5 Results of BRKGA-Random on SCP by instance group. 59
Table 4.6 Aggregated results of BRKGA-Random LS compared to BRKGA-Tuned LS for the three studied problems. 60
Table 4.7 Results of BRKGA-Random on FSP by instance group. 61
Table 4.8 Results of BRKGA-Tuned and BRKGA-Random on the FSP. 62
Table 4.9 Results of BRKGA-Random on TSP by instance group. 62
Table 4.10 Results of BRKGA-Tuned and BRKGA-Random on the TSP. 63
Table 4.11 Results of BRKGA-Random with local search on SCP by instance group. 64
Table 4.12 Results of BRKGA-Tuned and BRKGA-Random on the SCP. 64
Table 4.13 Output log of example case. BRKGA-Race execution on the FSP's instance TA10 for one hour. 66
Table 4.14 Parent configuration and resulting adapted configurations. 68
Table 4.15 Results of BRKGA-Race and BRKGA-Random on FSP, TSP, and SCP. 68
Table 4.16 Results of BRKGA-Race on FSP. 70
Table 4.17 Results of BRKGA-Race on TSP. 74
Table 4.18 Results of BRKGA-Race on SCP. 77

Table A.1 The table presents the dimensions for the FSP instances used in this work. 91
Table A.2 The table presents the dimensions for the SCP instances used in this work. 93

Table B.1 The table presents the complete results for the FSP without Local Search. Each row represents a problem instance, and “BKS” stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature. 96
Table B.2 The table presents the complete results for the FSP with Local Search. Each row represents a problem instance, and “BKS” stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature. 100

Table C.1 The table presents the complete results for the TSP without Local Search. Each row represents a problem instance, and “BKS” stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature. 105
Table C.2 The table presents the complete results for the TSP with Local Search. Each row represents a problem instance, and “BKS” stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature. 108

Table D.1 The table presents the complete results for the SCP without Local Search. Each row represents a problem instance, and “BKS” stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature. 112
Table D.2 The table presents the complete results for the SCP with Local Search. Each row represents a problem instance, and “BKS” stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature. 114

List of algorithms

Algorithm 1 Irace implementation by López-Ibáñez et al. [2]. 28

Algorithm 2 BRKGA with online random parameter values. 39

Unless the Lord builds the house, the builders labor in vain. Unless the Lord watches over the city, the guards stand watch in vain.

The Bible, *Psalms 127:1.*

1
Introduction

Metaheuristics present an alternative to traditional methods of mixed-integer optimization, especially when solving complex problems and/or large problem instances. They usually obtain good results in terms of solution quality and computing time. Metaheuristics work trying to find the best feasible solution to an optimization problem by evaluating potential solutions and performing iterative operations that seek to discover other (and possibly better) solutions Sörensen and Glover [4].

The Biased Random-Key Genetic Algorithm (BRKGA) [1] is a population-based metaheuristic, inspired by the process of natural evolution. This evolutionary algorithm (EA) applies the concept of survival of the fittest to obtain optimal or near-optimal solutions to combinatorial problems. To ensure the good performance of this algorithm (and other metaheuristics in general), defining parameter settings is a crucial step. Parameter values have a great influence on determining whether an optimal or near-optimal solution will be found by the algorithm and whether the search process will be efficiently run [5].

Parameter setting is also known as the algorithm configuration problem and it consists in finding parameter settings that optimize the empirical performance on a given set of problem instances [6]. It is impossible to obtain an optimal configuration of parameters that suits all problems. It is necessary to define parameter values for each implementation. Although, obtaining a configuration that leads to good results in a given set of instances does not guarantee that those values will be equally efficient in another set of instances of the same problem.

One way of tackling the parameter setting problem is through parameter tuning. Parameter tuning is the initialization of parameters in an *offline* manner. It consists of finding values for the parameters **before** the execution of the algorithm and fixing them throughout the algorithm's execution. The chosen setting is the one that presented the best results when applied to a certain set of problem instances. Offline tuning is a computationally intensive and time-consuming process. It is a task that has to be repeated whenever dealing with a different problem or different set of problem instances.

Another option is adopting the parameter control approach. This approach is also called *online* tuning and consists in dynamically defining parameters' values, along with the algorithm's execution. Parameter control is remarkably interesting when considering evolutionary algorithms due to the dynamic nature of EAs and their adaptive process [5]. Parameter control allows EAs to adapt parameter values according to different stages of the search process [7] and to accumulate information on the fitness landscape during the search to use this information in later stages. It also releases the user from the task of defining parameter settings, implicitly solving the tuning problem.

Indeed, for some applications, such as disaster aid, there is no available time to parametrization or similar problem instances to perform appropriate training. In that case, parameter control can help to reduce configuration time and provide settings that are suited for the problem instance at hand.

In this work, we seek to propose and evaluate parameter control approaches to BRKGA and compare them with the state-of-the-art approach to parameter tuning. As will be exposed in Section 2.2.1, the Iterated Race (irace) algorithm by López-Ibáñez et al. [2] is widely adopted within the scientific community to tune metaheuristics, including BRKGA (as seen in [8], [9], [10], [11], [12], [13], [14], [15], [16], [3], [17]). The algorithm is suitable for several metaheuristics and has a solid statistical foundation that supports its results. However, irace counts with several limitations. One of them is that the algorithm is very time-consuming and it is designed for scenarios where reducing computational time is not the primary objective.

Our first approach to eliminate the need of tuning BRKGA's parameters was adopting random parameter values for each of BRKGA's generations (i.e. iterations). With this idea, we sought to evaluate a simple concept before evolving to more sophisticated approaches. By implementing this method, we aimed to validate or refute the hypothesis that adopting random parameter values in BRKGA could lead to results as good as results obtained by the algorithm tuned with the current state-of-the-art approach in tuning – Iterated Race [2]. The computational experiments demonstrate that adopting random parameter values can be a promising method, having presented superior results on two of the three classical combinatorial optimization problems evaluated (that is, the Flowshop Permutation Problem and the Traveling Salesman Problem), when compared to the tuned algorithm.

After performing the first batch of experiments and observing good results of the random approach on some problems and not on others, we moved on to a more sophisticated approach that could lead to better results on the problems in which random parameters were not effective. With this in mind, our approach was to introduce the principles adopted by irace in BRKGA, designing an approach that could perform parameter control while solving optimization problems.

Throughout the execution of the experiments, we aimed to understand if varying the parameters' values along with the generations (iterations) of BRKGA would enhance the algorithm's performance or if it is better to leave the values fixed from the start. Also, by adopting online tuning, we seek to evaluate the impact of adopting a different set of parameter values for each instance and investigate if it leads to better results than using a fixed set of parameter values for the entire group of instances. By comparing the adopted methods, we can observe that varying parameters' values throughout the iterations of BRKGA is beneficial even when it is done randomly without including further knowledge in the parameters' adaptation.

In order to further discuss our proposals and their results, we structured this document as follows. In Chapter 2, we present the theoretical foundation for this research and related works. Chapter 3 describes the proposed methods and how BRKGA had to be adapted to incorporate its new features. We describe the studied hypotheses in Chapter 4, along with the experiments performed to evaluate them. Finally, in Chapter 5, we summarize the results and findings of this work, outlining future research possibilities.

2
Related Literature

This chapter presents the theoretical foundation for this work. It covers the description of the Biased Random-Key Genetic, along with some additional features proposed to the BRKGA framework and recent applications. We also detail the problem of algorithm configuration, describing both parameter tuning and parameter control approaches.

2.1
Biased Random-Key Genetic Algorithm

Genetic algorithms (GA) are search algorithms based on the mechanics of natural selection and genetics [18]. They are population-based metaheuristics, inspired by the process of natural evolution and a class of Evolutionary Algorithms (EA). While single-solution approaches move from a single point in the solution space to the next by applying some sort of transition function, GAs work from a collection of points simultaneously addressing different regions of the search space in parallel.

GAs require the encoding of solutions. Solutions are encoded by a string representation usually consisting of 0's or 1's, or some other finite alphabet. These solutions are called chromosomes, and the composing parts of the strings are called genes. A GA usually starts with a population of random chromosomes. Each chromosome is evaluated and given reproductive opportunities in a way that chromosomes that represent better solutions to the problem have more chances to generate offspring [19].

In the Biased Random-Key Genetic Algorithm (BRKGA) [1] chromosomes are represented as a vector of randomly generated real numbers between the interval $[0, 1]$, as proposed initially by Bean [20]. A deterministic algorithm called *decoder* associates chromosomes with solutions of the combinatorial optimization problem. The decoder also produces a *fitness* value, that represents the solution quality regarding the problem being considered.

A set of chromosomes forms a population that is evolved over a certain number of generations (algorithm iterations). In each generation, the decoder calculates the fitness of all individuals. The fitness is obtained by evaluating the solution by the objective function of the problem. Figure 2.1 illustrates the process of transitioning between one generation k to the next generation $k+1$. The population is divided into two groups: the elite group of individuals (those with the best fitness values) and the remaining group of non-elite

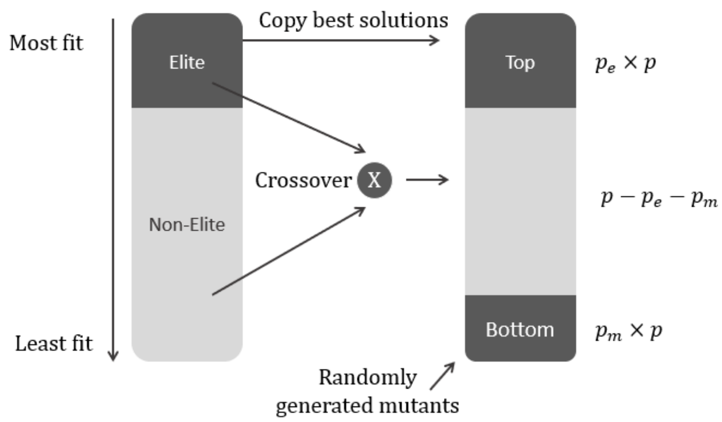

Figure 2.1: Transition between generations on BRKGA. Adapted from [1].

individuals. BRKGA adopts an elitist strategy since the elite population (a percentage of p_e of the total number of individuals) is fully migrated to the next generation. This results in a monotonically improving heuristic. Mutation occurs by introducing a fraction p_m of completely new chromosomes, called mutants, into the population of every generation. The remaining $p - p_e - p_m$ individuals to complete the population are generated through mating.

Mating happens by selecting one parent randomly from the elite set and another parent from the non-elite set (or from the entire population) [1] and performing a crossover operation that combines genes from both parents. The probability that an offspring inherits an allele from its elite parent is controlled by the parameter $\rho_e > 0.5$. Say we have $\rho_e = 0.7$. Then, the offspring will inherit the allele of the elite parent with probability 0.7 and of the other parent with probability $1-\rho_e = 0.3$. This way, it is more likely to inherit characteristics of the elite parent. The bias in BRKGA comes mostly from these differences in mating since it leads to elite individuals having a higher probability of passing on their characteristics to future generations. Adopting these elitist strategies supports the fast convergence and high-quality solutions [17].

2.1.1 Additional Features

Since the publication of Gonçalves and Resende [1] work, many authors proposed the addition of new features to the original BRKGA framework, to address concerns and/or to improve its efficiency.

As in GAs, premature convergence can be a concern in BRKGA. It happens when a population of chromosomes cannot produce offspring that outperform their parents and the population loses its diversity [21]. The latest variants of BRKGA have been introducing mechanisms to avoid this behavior.

That is the case of the work by Andrade et al. [3], that introduced the *shaking* procedure. To escape from local optima, a common approach in GAs is to reset the population (or restart the algorithm). In order to avoid destructing the convergence structure of the population by a full population reset, this article proposes a shaking. With the shaking feature, all individuals from the elite set suffer a perturbation and the remaining population is reset. The feature seeks to guarantee diversity in the non-elite set and preserve useful parts of solutions in the elite set. In this work, the use of the shaking procedure led to better solutions.

Andrade et al. [17] proposed BRKGA-MP-IPR, a new variant of BRKGA with the employment of multiple (biased) parents (MP) to generate offspring instead of the usual two, and hybridization with an implicit path-relinking local search procedure. By using multiple biased parents the authors seek to reinforce the bias in BRKGA, which is a key enabler of the success of the algorithm. In addition to multiple parents, the authors propose an implicit path-relinking (IPR) method. Path relinking is an intensification strategy that aims to exploit the intermediate solutions between two sufficiently diverse feasible solutions. Original implementations of this method operate explicitly on the solution neighborhood. The new implicit proposal allows the method to operate implicitly on the structure of the random-key vector, being more generic and modular. Results showed that both strategies lead to better solutions than those found by the standard BRKGA.

Ribeiro et al. [22] also presented a variation with path-relinking. The method was presented as a progressive crossover strategy to BRKGA. Path-relinking is applied to two-parent solutions to generate the best offspring that could be obtained by applying the standard crossover to those parents. Results of this work presented that the proposed approach is effective.

In GAs, it is possible to work with multiple populations at the same time. In a parallel GA, the algorithm work simultaneously on independent subpopulations. Periodically, these subpopulations communicate. Usually, this communication consists in exchanging individuals [23]. The idea behind a parallel GA is to avoid the propagation of local minimum solutions and to achieve good solutions faster. Some authors applied this idea in BRKGAs with multiple populations.

Gonçalves and Resende [24] presented a multi-population BRKGA in which three populations are evolved independently in parallel and after a pre-determined number of generations, the overall two best chromosomes (from the union of all populations) are inserted into all populations.

De Faria et al. [25] employed four parallel populations evolving independently and periodically exchanging good quality solutions to solve an electric distribution network reconfiguration problem. As in [24], the two best chromosomes from all populations are inserted into all the other populations after a certain number of generations.

Amaro et al. [26] proposed considering μ populations $(P_1, P_2, ..., P_\mu)$ and, after a number of generations, the whole elite set of P_μ is inserted into $P_{\mu+1}$ (as P being a circular list). Alixandre and Dorn [27] proposed a Distributed BRKGA (D-BRKGA), considering a different exchange strategy. The authors applied a stratified migration policy that randomly selected 10% of the individuals of the elite set, the non-elite set, and mutants to migrate between populations.

Oliveira et al. [28] proposed a co-evolutionary algorithm for solution and scenario generation in stochastic problems based on BRKGA. In this work, BRKGA works with two populations with different ends: one *solution* population and one *scenario* population. The fitness of solutions depends on how they perform in the face of the scenarios in the scenario population.

Additional features to avoid premature convergence that are related to parameter calibration are of particular interest to this work. However, for organization purposes, these will be explored in Section 2.2.

2.1.2 Applications

In the past few years, several applications of BRKGA have been addressed in the literature. The method is vastly applied in strategic-level planning, in problems such as facilities location and network design, and tactical and operational planning, as in scheduling and vehicle routing problems. The method has been successful in dealing with complex problems and large instances.

Within the applications of BRKGA, we can mention Mauri et al. [12], that applied a hybrid approach combining BRKGA with a clustering search in order to minimize total costs of the multiproduct two-stage capacitated facility location problem where a set of different products must be transported from a set of plants to a set of intermediate depots and from these depots to a set of customers. Biajoli et al. [29] tackled the single product version of this same problem while combining BRKGA with a new local search for the TSCFLP. Londe et al. [13] addressed the p-next center problem applying BRKGA combined with different local search proposals. Stefanello et al. [8] considered the placement of virtual machines across multiple data centers, meeting the quality of service requirements while minimizing the bandwidth cost of the data centers. The authors compared the use of a greedy randomized adaptive search procedure and a biased random-key genetic algorithm, both hybridized with a path-relinking strategy and a local search.

Gonçalves and Resende [30] applied a hybrid approach with BRKGA and a linear programming model to address the unequal area facility layout problem, seeking to determine the order of placement, dimensions, and position of each facility. Andrade et al. [14] introduced the wireless backhaul network design problem (a problem closely related to variants of the Steiner tree problem and the facility location problem) motivated by the requirements of real-world telecommunication networks and addressed it with BRKGA. Lalla-Ruiz et al. [31] used a hybrid approach of BRKGA and a local search to solve the Quadratic Assignment Problem (QAP). Pinto et al. [15] provided a hybridization of a BRKGA with an exact local search strategy to tackle the maximum quasi-clique problem. We can also mention Pessoa et al. [32] for the application of BRKGA to address the tree of hubs location problem. In the field of Machine Learning, Cicek et al. [33] sought to determine the design and weight parameters of Artificial Neural Networks with BRKGA.

Considering problems more related to tactical and operational planning, we can point out other applications of BRKGA, demonstrating the algorithm's relevance to the industry in general. That is the case of Carrabs [34] that applied BRKGA combined with local search to address the set orienteering problem where customers are grouped in clusters, and the profit associated with each cluster is collected by visiting at least one of the customers in the respective cluster. Also, Abreu et al. [9] addressed open shop scheduling with routing by capacitated vehicles using BRKGA with an iterated greedy local search procedure. Kummer et al. [10] applied BRKGA to the Vehicle Routing Problem with Time Windows and Synchronization Constraints and outperformed the previous best-known solutions found by up to 25%, using less than half of the computational times reported previously.

The single- and multi-round divisible load problem is addressed in Ribeiro et al. [22], with the BRKGA variant with path-relinking as a progressive crossover strategy. This problem consists of the distribution of computational work among different processors to be treated in parallel. The work of Andrade et al. [3] approached the permutation flow shop scheduling problem with total flowtime minimization with BRKGA with the shaking procedure. In this problem, one considers a set of jobs to be scheduled on a set of machines. Each job has a processing time on each machine and can be executed on only one machine at a time. De Faria et al. [25] applied their multi-population variant of BRKGA to the electric distribution network reconfiguration problem. In this problem, the topology of the distribution system is modified in order to reduce power losses on the feeders. Pessoa and Andrade [16] applied BRKGA to the flow shop scheduling problem with delivery dates and cumulative payoffs. This problem is a variation of the flow shop scheduling problem considering job release dates and aims to maximize the total payoff with a stepwise job objective function. In this work, when compared to other metaheuristics (ILS and IGS), BRKGA led to superior results.

The BRKGA-MP-IPR variant [17] was applied to three real-life scenarios. The first one is the wireless backhaul network design problem. In this problem, a set of demand points must be addressed by small cells (radio base stations) that can be connected to a set of root points either by fiber or wireless links. The second application is the firmware-over-the-air scheduling problem. In this problem, a schedule for connected cars to initiate a download/update session over LTE networks must be created. The last problem treated in this article is the Winner Determination Problem (WDP). This problem represents a combinatorial auction in which a seller should pick a set of non-overlapping bids to maximize the total selling value.

Amaro et al. [26] implements their proposal of a parallel BRKGA to the irregular strip packing problem (ISPP). This problem is a class of cutting and packing problems in which a set of items with arbitrary dimensions and shapes must be placed in a container with a variable length. Gonçalves and Resende [24] applied the multi-population BRKGA to the single container loading problem where several rectangular boxes of different sizes are loaded into a single rectangular container.

Cunha et al. [11] considered the Rescue Unit Allocation and Scheduling Problem (that can be seen as a generalization of the unrelated parallel machine scheduling problem with sequence and machine-dependent setup), addressing it with BRKGA. Zudio et al. [35] addressed the Three-dimensional Bin Packing Problem with a hybridization of BRKGA and a variable neighborhood descent-inspired algorithm. The Capacitated Vehicle Routing Problem with Time Windows was explored by Rochman et al. [36], which applied a modified BRKGA that considered chromosomes' gender. Damm et al. [37] applied BRKGA to the field technician scheduling problem. Chaves et al. [38] addressed the minimization of tool switches problem using a hybrid version of BRKGA with cluster search. Amaro and Pinheiro [39] addressed a special class of cutting and packing problems called Nesting Problems with a parallel biased random-key genetic algorithm with multiple populations. Gonçalves [40] handled a very common problem in the home textile industry, the production, and cutting problem.

As seen in the literature review above, BRKGA is suited for many industries' relevant applications and can be tailored to meet the needs of different contexts. However, the performance of the algorithm highly depends on the configuration of parameters that will control the evolution process. In the basic version of BRKGA [1], these parameters are: population size (p), proportion of elite individuals (p_e), proportion of mutant individuals (p_m), and probability of inheriting a gene from parents from the elite set (ρ_e). Considering multiple populations, it is also needed to set the number of populations and the information exchange rate. With additional features, like in newer versions such as the BRKGA-MP-IPR variant [17], the number of control parameters is even higher. In the next section, we will address the problem of parameter settings.

2.2 Parameter Settings

Parameter setting is also known as the algorithm configuration problem. It consists in "finding parameter settings (or configuration) for which the empirical performance on a given set of problem instances is optimized" [6]. From a machine learning point of view, parameter configuration can be considered a learning problem, in which one seeks to obtain a good parameter setting to solve unknown instances from learning in a set of training instances [41]. Hoos [6] states the problem as follows:

> Given
>
> – an algorithm A with parameters $p_1, ..., p_k$
> – a space C of configurations, where a configuration $c \in C$ defines values for A's parameters
> – a set of problem instances I
> – a performance metric m that measures the performance of A on instance I
>
> find a configuration $c^* \in C$ that results in optimal performance of A on I according to metric m.

The number and types of parameters influence this problem's complexity. Usually, the parameter configuration must not only perform well on set I of problem instances but also in unknown problem instances. When a problem presents different instance types, the difficulty of finding a good configuration rises [6]. Also, in attempting to obtain the best possible set of values, it is necessary to consider the interactions between parameters and evaluate configurations as a whole. According to Eiben et al. [42], parameter configuration can be done before or during the execution of the algorithm. When done before the execution, it is called **parameter tuning** or offline tuning. When done during the execution, it is called **parameter control** or online tuning.

2.2.1 Parameter Tuning

Initializing parameters *offline* consists of finding values for the parameters **before** the execution of the algorithm and fixing them throughout the algorithm's execution. The chosen setting is the one that presented the best results when applied to a certain set of problem instances. There are usually two phases in parameter tuning: the configuration phase (tuning) and the production phase. In the configuration phase, the aim is to optimize the parameter values based on the training instances selected to represent the problem. In the production phase, the obtained configuration is applied to new instances.

This type of tuning is widely adopted for configuring metaheuristics [42]. Eiben et al. [42] pointed out that it was common to define parameter values manually until the moment of their publication. Different values were tested, and those with the best results were adopted. A few decades later, Huang et al. [41] highlighted the increasing demand for systematic and automated approaches to parameter setting as problems and solution approaches became more complex.

One possible simple way to tackle parameter tuning is via a grid search approach, also known as the full design of experiments. In grid search, all possible combinations of given discrete parameters are evaluated. It is a straightforward method but computationally intensive. Depending on the dimensionality of the configuration space the computational complexity may increase, growing exponentially and making this task impossible to perform [43]. Depending on the number of parameters to be defined, it is a difficult task even for individual optimization of the parameters, disregarding their interactions and co-dependence [5].

Another possibility is to adopt parameter values as suggested for similar groups of problem instances in the literature. As pointed out by Karimi et al. [44], the No Free Lunch Theorem [45] reports that the performance of a particular algorithm with a specific parameter configuration on a few sample problem instances are of limited utility. They warn that one should be cautious when generalizing those results to other problem instances, because there are no guarantees that the configuration will perform equally well.

Among offline automated methods of parameterization are F-Race [46], CALIBRA [47], Iterated F-Race [48], Meta-EAs [49], ParamILS [50] and others.

These methods have different approaches with different levels of complexity and may include the use of heuristic searches, and statistical techniques, among others.

Considering our research on BRKGA, the most frequent approaches for parameter tuning are the adoption of values suggested by the literature, the use of grid search (trying different combinations of parameter values), and the irace [2] method. CALIBRA [47] also appeared in Biajoli et al. [29] work. In Table 2.1, we can see the parameter tuning methods employed in the articles referred to in Section 2.1.

Table 2.1: Parameter tuning methods adopted for BRKGA's configuration.

Literature Suggestion	**Grid Search**	**irace**	**CALIBRA**
Chaves et al. [38]	Carrabs [34]	Stefanello et al. [8]	Biajoli et al. [29]
Cicek et al. [33]	Rochman et al. [36]	Abreu et al. [9]	
De Faria et al. [25]	Damm et al. [37]	Kummer et al. [10]	
Amaro and Pinheiro [39]	Gonçalves [40]	Cunha et al. [11]	
Amaro et al. [26]	Alixandre and Dorn [27]	Mauri et al. [12]	
Oliveira et al. [28]	Gonçalves and Resende [24]	Londe et al. [13]	
Gonçalves and Resende [30]	Lalla-Ruiz et al. [31]	Andrade et al. [14]	
Ribeiro et al. [22]	Pessoa et al. [32]	Pinto et al. [15]	
Zudio et al. [35]		Pessoa and Andrade [16]	
		Andrade et al. [3]	
		Andrade et al. [17]	

When using a tuning method that tests possible parameter values (such as grid search, irace, or CALIBRA) it is necessary to provide an interval for the parameter values to be chosen from. In most articles accessed in our research ([10], [39], [33], [35], [12], [8], [9], [34], [38], [30], [11], [14], [24], [40], [29], [32], [13], [36], [22], [25], [3], [17]), the chosen interval is based on the suggestion of Gonçalves and Resende [1]. The suggested parameter ranges are described in Table 2.2. Variations are most frequently observed in the population size or mutants percentage. The population size is sensible to the problem size. A large problem with a large population may be too computational expensive and lead to too few iterations of the algorithm in a predetermined time. Regarding the mutants percentage, depending of the problem tendency to premature convergence, a higher value might be beneficial.

Table 2.2: Recommended parameter ranges by Gonçalves and Resende [1].

Parameter	**Description**	**Recommended Range**
p	size of population	$p = an$, where $1 \leq a \epsilon$ R is a constant and n is the chromosome length
p_e	size of elite population	$0.10p \leq p_e \leq 0.25p$
p_m	size of mutant population	$0.10p \leq p_m \leq 0.25p$
ρ_e	elite allele inheritance probability	$0.50p \leq \rho_e \leq 0.80p$

2.2.1.1 Iterated Race

Among the most used parameter tuning approaches to BRKGA is Iterated Race (irace). Irace [2] is an implementation of a general iterated racing procedure, which includes I/F-Race [48] as a special case that includes Friedman's nonparametric two-way analysis of variance by ranks. I/F-Race consists of mainly three steps, as can be seen in Figure 2.2. The following steps are repeated until a stop criterion is met: sampling new configurations according to a particular probability distribution, selecting the best configurations by means of *racing*, and updating the sampling distribution biasing them toward the best configurations.

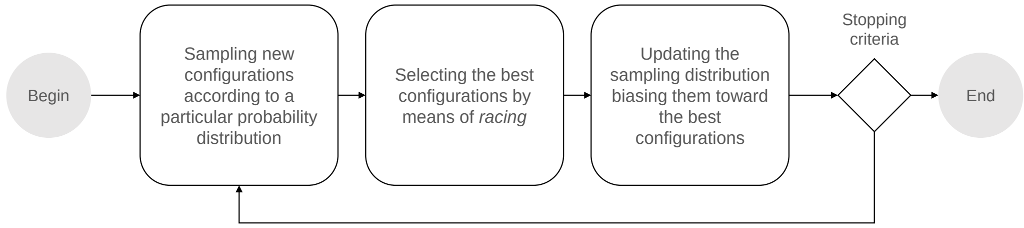

Figure 2.2: Main steps performed by irace. Adapted from [2].

The algorithm begins by sampling new configurations using a sampling distribution associated with each parameter. Irace [2] uses a truncated normal distribution for numerical parameters and a discrete distribution for categorical parameters, while ordinal parameters are treated as in the numerical case. The algorithm seeks to bias these distributions along with the iterations in order to increase the probability of sampling the parameter values of the best configurations found. Every time the algorithm has to update these distributions, it does so by modifying the mean and the standard deviation of the normal distribution or the discrete probability values of the discrete distributions.

The best configurations are selected by racing, as illustrated in Figure 2.3. A *race* starts with a finite set of candidate configurations. At each step of the race, the candidate configurations are evaluated on a single instance. After a few steps, the candidate configurations that perform statistically worse than at least another one are discarded, and the other configurations (the surviving ones) remain in the race. The first statistical test is only performed after a high number of instances are seen, given that the first test is crucial in the elimination of configurations. The following tests are done more frequently. The procedure continues until reaching a determined computational budget (defined as maximum time or a number of experiments), or reaching a minimum number of surviving configurations.

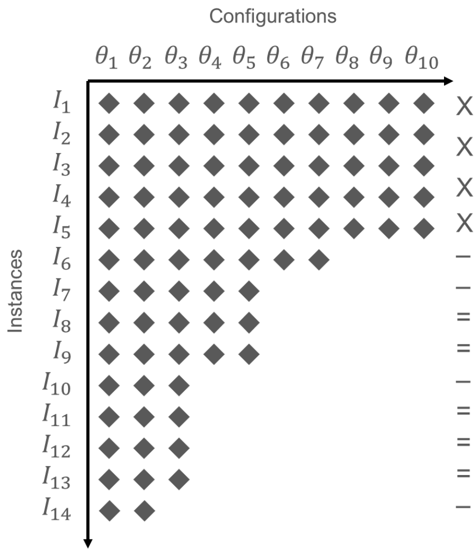

Figure 2.3: A race illustration. The rows represent the instances, and the columns represent the configurations. Each node is an evaluation of one configuration on one instance. On the right, "X" indicates that no statistical test was performed, "–" shows that the test eliminated at least one configuration, and "=" indicates that the test did not discard any configuration. Adapted from López-Ibáñez et al. [2].

The authors López-Ibáñez et al. [2] define the algorithm implemented in irace as "a search process based on updating sampling distributions" where the main element is the combination of a search process with an evaluation procedure that considers the stochasticity of the evaluation. In this work, we aim to embrace this principle while incorporating the proposed methodology inspired in irace into BRKGA.

Irace is available as an R package. A user guide on the package can be found in [51]. The algorithm implementation is described in Algorithm 1. As for input data, Irace requires a set of instances $\mathcal{I}$, a parameter space X, a cost function $\mathcal{C}$, and a tuning budget B.

Some definitions that must be made before running the algorithm are described in [2]. First, irace defines how many races N^{iter} (or iterations) will be executed. The authors suggest that this number is a function of the number of parameters, being defined as $N^{iter} = \lfloor 2 + \log_2 N^{param} \rfloor$. It allows that for larger parameter spaces, more iterations are run. The idea is that configurations generated in later iterations will be more similar and more iterations will be needed to identify the best ones.

Each race has a computation budget $B_j = (B - B^{used})/(N^{iter} - j + 1)$, where $j = 1, ..., N^{iter}$. Each race evaluates a set of configurations Θ_j, that

Algorithm 1: Irace implementation by López-Ibáñez et al. [2].

Data: $I = \{I_1, I_2, ...\} \sim \mathcal{I}$,
parameter space X,
cost measure $\mathcal{C}(\theta, i) \in \mathbb{R}$,
tuning budget B

1 $\Theta_1 \leftarrow \text{SampleUniform}(X)$;
2 $\Theta^{elite} \leftarrow \text{Race}(\Theta_1, B_1)$;
3 $j \leftarrow 1$;
4 **while** $B^{used} \leq B$ **do**
5 $\quad j \leftarrow j + 1$;
6 $\quad \Theta^{new} \leftarrow \text{Sample}(X, \Theta^{elite})$;
7 $\quad \Theta_j \leftarrow \Theta^{new} \cup \Theta^{elite}$;
8 $\quad \Theta^{elite} \leftarrow \text{Race}(\Theta_j, B_j)$;
9 **return** Θ^{elite};

is calculated as $|\Theta_j| = N_j = \lfloor B_j/(\mu + T^{each} \cdot \min\{5, j\}) \rfloor$. The parameter μ is equal to the number of instances needed to perform the first statistical test ($\mu = T^{first}$) and T^{each} is the interval in which subsequent statistical tests are performed. In the default settings of the irace package, $T^{each} = 1$. With this definition, Θ_j decreases with the number of iterations, allowing more evaluations per configuration to happen in later iterations. It also keeps the algorithm from decreasing N_j beyond the fifth iteration, to avoid having too few configurations to be evaluated in a single race.

After defining these parameter values, irace [2] samples the initial set of candidate configurations by uniformly sampling the parameter space X (line 2 of Algorithm 1). Each configuration is evaluated on the first instance by observing the cost measure $\mathcal{C}$. Configurations are iteratively evaluated on the following instances until T^{first} instances are seen. After there is a relevant set of data on each configuration, a statistical test is done on the results. If a configuration performed worse than at least another configuration, it is removed from the race. The surviving configurations stay in the race and are evaluated in the next instance.

To select which configurations are discarded, irace uses the non-parametric Friedman's two-way analysis of variance by ranks (the Friedman test [52]) by default. Other tests are available within the package, such as the t-test. The authors López-Ibáñez et al. [2] indicate that the choice of the test that will be performed depends on the chosen cost measure $\mathcal{C}$. When the cost function for different instances is not commensurable or the tuning objective is an order statistic (such as median), the Friedman test is more appropriate. Irace uses the statistical tests as a selection heuristic and the statistical significance level is not preserved when it cuts search performance.

A race continues until the budget of the current iteration is no longer sufficient to evaluate all remaining candidate configurations on a new instance, or when at most the minimum number of configurations remains. At the end of a race, the elite set of configurations Θ^{elite} is selected to survive to the next race. This set is defined by first assigning a rank r_z according to the cost measure observed on the evaluations. The $N_j^{elite} = min\{N_j^{surv}, N^{min}\}$ configurations with the lowest rank compose the elite set. Before starting the next race, a number of $N_j^{new} = N_j - N_{j-1}^{elite}$ new candidate configurations are generated (line 7 of Algorithm 1). Then, in the following race $N_j^{new} + N_{j-1}^{elite}$ configurations are evaluated.

To generate a new configuration, the following procedure is executed by irace [2]. First, one parent configuration θ_z is sampled from the set of elite configurations with a probability proportional to rank r_z. By doing so, "higher-ranked configurations have a higher probability of being selected as parents" [2]. Then, a new value is sampled for each parameter. Consider that X_d is a numerical parameter defined within the range $[\underline{x}_d, \bar{x}_d]$. To obtain a new value, irace samples it from the truncated normal distribution $\mathcal{N}(x_d^z, (\sigma_d^j)^2)$. The mean of the distribution x_d^z assumes the value of parameter d in parent configuration θ_z. The standard deviation σ_d^j is set to $(\underline{x}_d - \bar{x}_d)/2$ initially, and then decreased at each iteration following Equation (2-1). This allows the sampled values are increasingly closer to the value of the parent configuration, intensifying the search around the best parameter settings found.

$$\sigma_d^j = \sigma_d^{j-1} \cdot \Big(\frac{1}{N_j^{new}}\Big)^{1/N^{param}} \tag{2-1}$$

After adapting the distributions, the new configurations are sampled. A set with the newly generated configurations and the elite configurations is generated (as seen in line 8 of Algorithm 1) and a new race starts (line 9). If the budget is exhausted the algorithm stops.

The authors López-Ibáñez et al. [2] point out that the parameters of the algorithm must be fine-tuned in order to obtain the best performance. When stating some limitations of irace, the authors indicate that the algorithm is time-consuming and it is designed for scenarios where reducing computational time is not the primary objective. Also, when providing too small tuning budget to irace, the resulting configuration might not be better than a random setting.

2.2.2 Parameter Control

Parameter control or *online* tuning consists in dynamically defining parameter values, along with the algorithm's execution. It is an alternative to offline tuning, as it starts with a set of values that are updated during the optimization process. Sevaux et al. [53] define a classification for parameter control approaches, applicable to any metaheuristics. For the authors, *adaptive metaheuristics* include mechanisms to modify their configuration during execution and *multilevel metaheuristics* use other metaheuristics to adjust their configuration.

Parameter control saves resources by eliminating the need to tune the parameters before starting the optimization process. As stated earlier, in parameter tuning, several combinations of parameter values are tested and the search space is explored for a considerable amount of time to evaluate configurations. During this process, a lot can be learned about the solution space and contribute to finding good solutions faster. Usually, these findings are wasted, not being observed and used in the actual optimization problem. In addition, updating parameter values during the search process can provide an adequate balance between diversification and intensification. Considering evolutionary algorithms, parameter control is particularly interesting due to the dynamic nature of EAs and their adaptive process [5].

As pointed out by Karafotias et al. [7], parameter control allows EAs to adapt parameter values according to different stages of the search process. Also, allows EAs to accumulate information on the fitness landscape during the search and use this information in later stages. Furthermore, parameter control releases the user from the task of defining parameter settings, implicitly solving the tuning problem. Even if the parameter choice is still needed, it can be hidden behind design decisions.

Eiben et al. [42] proposes a classification of parameter control based on evolutionary algorithms. Parameter control can happen in a *deterministic*, *adaptive*, or *self-adaptive* way. In a deterministic approach, parameter values are updated according to fixed and predetermined rules. A common approach is to define a rule as a function of time, without considering information on the current state of the search, according to Eiben et al. [42]. This approach already presents advantages over static parameters. The adaptive approach consists of using some research feedback as input for a mechanism that determines the update magnitude or direction in the parameter values. Finally, in the self-adaptive approach, the parameters are encoded in chromosomes and undergo mutation and recombination. The best parameter values lead to better individuals, who tend to survive and propagate their parameters setting.

One approach to parameter control can be seen in the genetic algorithm with variable population size [54]. In this work, the authors proposed that population was not an adjustable parameter, but a measure derived from the stage of evolution. Individuals are given a lifespan when they are created and their lifespan is reduced every generation until they are removed from the population. The lifetime of each individual depends on their *fitness* value, thus, individuals who represent better solutions have a longer lifetime and can generate more offspring. This work present good preliminary results that encourage more research on the topic.

In Hinterding et al. [55], the proposed algorithm (Self-Adaptive Genetic Algorithm or SAGA) presents the population size and mutation intensity as self-adaptive parameters. SAGA builds on the concept of co-evolution for population adaptation by defining a community with three genetic algorithms with different population sizes. The best solutions found at the end of a number of generations are used to modify population sizes, taking into account upper and lower limits. This work shows that self-adapting more than one parameter of GAs is both possible and beneficial.

Bäck et al. [56] seek to eliminate three parameters from the genetic algorithm – population size, mutation rate, and crossing rate, considered by the author as the main parameters responsible for the evolution strategy. For the population size, the adopted approach resembles the proposal of [54]. Individuals are given a lifespan that decays according to a bi-linear function. One of the differences in this proposal is that the individual with the best *fitness* value does not have their lifespan reduced in each generation. The proposed algorithm is compared to the traditional algorithm, and the results showed that population size adaptation was crucial for the observed improvements. The algorithm with the adaptive population gave almost as good results as the algorithm with all the adaptive parameters. Thus, the author highlights the importance of studying control mechanisms, especially for population size.

Seeking to investigate the effect of variable population size, Eiben et al. [57] introduce a new population scaling mechanism. In their proposal, when there is an improvement in *fitness*, the algorithm tends to explore space for solutions, and this occurs through the increase in population size. When there is a short period without improvement, the population decreases. In this phase of decline, it is expected that the search will intensify, due to the reduction in diversity in the population. If this period of stagnation becomes too long, the population increases again to encourage exploration. The results confirm that population size adaptation brings advantages for the execution of genetic algorithms, mainly in terms of the efficiency of the algorithm, which can execute more generations and, consequently, achieve better results.

Aleti et al. [58] studies another approach to EAs parameter control. They examine suitability of several time series prediction methods to project the probabilities to use for parameter value selection based on previous data. Their study indicates that prediction methods can be applied, specially for EAs, since all standard parameters with the exception of population size conform to prediction methods assumptions (such as linearity, normality of the error distribution, homoscedasticity, etc.).

There are also recent movements toward parameter control in BRKGA. We can see it in Chaves et al. [59] that proposed an adaptive version of the algorithm (A-BRKGA). The parameters population size, proportion of elite and mutant individuals, probability of inheriting a gene from the father of the elite set, and a maximum number of generations are updated according to deterministic rules that consider the progress of evolution. Two self-adaptive parameters are introduced, α and β, which evolve along with the search. At the beginning of the evolutionary process, the population size receives the maximum value and decreases throughout the process, based on the γ parameter which is chosen by the user based on three predetermined values. Each value of γ allows the population size to decay at a different rate until reaching the minimum value. The maximum number of generations is used as stopping criteria and can take specific values based on the parameter γ. Note that A-BRKGA removes the p parameter but introduces three other parameters (α, β, and γ) which must be set offline. Results show that A-BRKGA performed as well as BRKGA in terms of solution quality for the capacitated clustering center problem. In that work, the authors applied the proposed method to the Capacitated Centered Clustering Problem and, when compared to BRKGA, A-BRKGA presented similar robustness and computational time.

Chaves et al. [60] proposed a Reinforcement Learning approach to parameter control in BRKGA. Reinforcement Learning is a Machine Learning field where an agent interacts with the environment and takes the actions that maximize the reward. The Q-Learning method controls BRKGA parameters (the population size, proportion of elite and mutant individuals, the probability of inheriting a gene from the elite set's parent) and the parameters of the method itself (ϵ, learning factor, discount factor) based on pre-determined values. A value for each parameter is chosen using a greedy policy. If the current configuration shows improvement in the best chromosome in the population, a reward is generated and the learning function is updated. In this work, the authors applied the proposed method to the Traveling Salesman Problem. Also related to machine learning techniques, in the work by Schuetz et al. [61] the optimization for the BRKGA parameters is done using Bayesian optimization techniques in an online manner - the HOA method (Automatic Hyperparameter Optimization) described in Bergstra et al. [62]. In this paper, BRKGA is applied to optimize robot trajectory planning at industry-relevant scales.

2.3 Concluding Remarks

During the literature review, we assessed recent works on BRKGA and parameter control methods. Evidence demonstrates that BRKGA is a strong algorithm with fast convergence and higher-quality solutions, especially when compared to the standard RKGA. The method's applications surveyed in this work illustrate its applicability in complex problems, as summarized in Table 2.3. Still, a common concern when implementing GAs is the possibility of premature convergence. Some of the approaches presented in the literature may be using multi-population algorithms to avoid the propagation of local minimums and achieve good solutions faster. Also, calibrating parameters properly supports avoiding this matter.

BRKGA, like GAs and other metaheuristics, highly depends on adequate parameter values in order to achieve its potential. Most of the surveyed applications utilize some form of parameter tuning, as summarized in Table 2.3. However, offline tuning is a computationally intensive and time-consuming process. A task that has to be repeated whenever dealing with a different problem or different set of problem instances. Our research shows that this step is present in most applications of BRKGA, and can impact directly its performance. This step can present itself as an obstacle to the application of a metaheuristic framework when dealing with real-life problems, especially those without a proper training set, or with time constraints.

Online tuning, on the other hand, allows the parameter setting task to be performed along with the algorithm execution, saving time and resources. Considering EAs is especially interesting due to the dynamic nature of the evolution process. It allows different parameter values for different stages of the evolution and is completely suited to the instances at hand. Even though there is plenty of literature addressing the parameter control approach on GAs, little work has been done in BRKGA regarding this approach.

By studying the related literature on these matters, we noticed that advances in parameter control in BRKGA are still relatively incipient, with only two published papers of our knowledge. In this context, this work seeks to advance the state-of-art of parameter control in BRKGA, evaluating two proposals for adapting its parameter values during the algorithm's execution. We aim to focus on two approaches not yet explored in BRKGA: the adoption of random parameter values for each of the algorithm's generations, and the incorporation of irace's learning mechanism in an online manner, integrating it into BRKGA's framework. By doing so, we aim to contribute to providing problem-solving frameworks that are highly adaptive to different problems and problem instances, being able to tune themselves while solving problems efficiently.

Table 2.3: Summary of BRKGA applications and parameter settings methodologies.

Article	**Application**	**Parameter Setting Methodology**
Chaves et al. [60]	Travelling Salesman Problem	Adaptive
Chaves et al. [59]	Capacitated Centered Clustering Problem	Adaptive
Schuetz et al. [61]	Robot Trajectory Planning	Automatic Hyperparameter Optimization (HOA)
Biajoli et al. [29]	Two-Stage Capacitated Facility Location Problem	CALIBRA
Alixandre and Dorn [27]	CF3 and CF4 Functions	Grid search
Carrabs [34]	Set Orienteering Problem	Grid search
Damm et al. [37]	Field Technician Scheduling Problem	Grid search
Gonçalves and Resende [24]	3D Single Container Loading Problem	Grid search
Gonçalves [40]	Production and Cutting Problem	Grid search
Lalla-Ruiz et al. [31]	Quadratic Assignment Problem	Grid search
Pessoa et al. [32]	Tree Of Hubs Location Problem	Grid search
Rochman et al. [36]	Capacitated Vehicle Routing Problem with Time Windows	Grid search
Abreu et al. [9]	Open Shop Scheduling with Routing by Capacitated Vehicles	IRACE
Cunha et al. [11]	Rescue Unit Allocation and Scheduling Problem	IRACE
Andrade et al. [14]	Wireless Backhaul Network Design	IRACE
Andrade et al. [3]	Permutation Flowshop Scheduling Problem with Total Flowtime Minimization	IRACE
Andrade et al. [17]	Over-The-Air Software Upgrade Scheduling, Network Design Problems, and Combinatorial Auctions	IRACE
Cunha et al. [11]	Rescue Unit Allocation and Scheduling Problem	IRACE
Pessoa and Andrade [16]	Flowshop Scheduling Problem with Delivery Dates and Cumulative Payoffs	IRACE
Kummer et al. [10]	Vehicle Routing Problem with Time Windows and Synchronization Constraints	IRACE
Londe et al. [13]	P-Next Center Problem	IRACE
Mauri et al. [12]	Multiproduct Two-Stage Capacitated Facility Location Problem	IRACE
Pinto et al. [15]	Maximum Quasi-Clique Problem	IRACE
Stefanello et al. [8]	Virtual Machines Placement Across Multiple Data Centers	IRACE
Amaro and Pinheiro [39]	Nesting Problem	Literature suggestion
Amaro et al. [26]	Irregular Strip Packing Problem (ISPP)	Literature suggestion
Chaves et al. [38]	Minimization of Tool Switches Problem	Literature suggestion
Cicek et al. [33]	Determine the Design and Weight Parameters of Artificial Neural Networks	Literature suggestion
De Faria et al. [25]	Electric Distribution Network Reconfiguration Problem	Literature suggestion
Gonçalves and Resende [30]	Unequal Area Facility Layout Problem	Literature suggestion
Oliveira et al. [28]	General stochastic optimization function	Literature suggestion
Zudio et al. [35]	Three-Dimensional Bin Packing Problem	Literature suggestion
Ribeiro et al. [22]	Single- and Multi-Round Divisible Load Scheduling Problem	Literature Suggestion

3
Proposed Methods

In this chapter, we present the proposed methods for controlling parameter values of the Biased Random-Key Genetic Algorithm (BRKGA). Our first approach was adopting random parameter values for every generation of BRKGA. More details on how this method was implemented are described in Section 3.1.

The second approach was to introduce the tuning concepts observed in irace [2] to BRKGA, proposing a metaheuristic that tunes itself while solving the optimization problem. Details on the implementation are available in Section 3.2.

3.1
Random Parameter Values

Our first approach to eliminate the need for tuning BRKGA parameters is adopting random values for the parameters at each algorithm's generation. To our knowledge, this is the simplest possible approach to address parameter control on BRKGA. Therefore, following Occam's razor principle [63], we opted to evaluate the simplest hypothesis before proposing more complex approaches.

One may ask "if the parameter values are randomly selected, why not fix them throughout the execution?". One reason is that if we are unlucky and draw bad values at the beginning, we necessarily get a bad run. Therefore, randomly selecting different values at each generation balances the influence of "bad luck." Another reason is that sampling random values allows us to have a chance of selecting good parameter values in different stages of evolution. There is a higher chance that we will adopt adequate mutation and elite-parent inheritance rates when needed, than if we had fixed a value.

We assume random parameter values for the four main BRKGA parameters. These are the population size (p), the proportion of elite individuals (p_e), the proportion of mutant individuals (p_m), and the probability of inheriting a gene from parents from the elite set (ρ_e). It is given an interval (upper and lower bounds) for the values of each parameter and new parameter values are uniformly sampled within the provided intervals for every iteration of the BRKGA.

The description of the proposed method can be seen in Algorithm 2. First, we initialize the algorithm and randomly sample the parameter values (lines 1-3) from a uniform probability distribution. Each parameter has its own distribution $U(a, b)$ where a is the upper bound value for that parameter, and b is the lower bound value provided. The following steps consist of the standard BRKGA procedure that was introduced in Gonçalves and Resende [1]. In the first generation (k = 1), the population is initialized with random individuals (line 4), then we evaluate the fitness and sort the population (lines 5-6). The population is split into elite and non-elite sets and the elite partition is copied to the next generation (lines 16-17). Mutants are inserted and the mating procedure occurs for the remaining individuals of the population (lines 18-19).

At the beginning of each of the following generations (k > 1), we update the parameter values by randomly sampling values within the specified range (line 9), from their uniform distribution. This means that each value has the same probability of being drawn in every generation. Some considerations must be made regarding the population size. When changing the population size along with the generations, it is necessary to create or eliminate individuals from the current population. Consider the new population size p^k sampled for the current k generation, and p^{k-1} the population size of the previous generation. When $p^k > p^{k-1}$, $(p^k - p^{k-1})$ chromosomes are randomly generated and inserted into the population (lines 10-13). When $p^k < p^{k-1}$, the $(p^{k-1} - p^k)$ chromosomes with the least fitness values are removed from the population (lines 14-15). The elite and mutant partitions are recreated according to p^k. After the population management step, we follow the standard BRKGA procedure and update the current best solution when necessary (lines 20-22).

Algorithm 2: BRKGA with online random parameter values.

Data: Stop criteria.

Result: Best solution.

initialize best solution $B^* \leftarrow \infty$ // minimization problems;
initialize generations counter $k \leftarrow 1$;
uniformly sample random values for each parameter $(p^k, p_e^k, p_m^k, \rho_e^k)$;
initialize population P with p^k random individuals;
evaluate fitness values of individuals;
sort current population by fitness value;
while *a stopping criterion is not met* **do**
    if $k > 1$ **then**
        uniformly sample new values for each parameter $(p^k, p_e^k, p_m^k, \rho_e^k)$;
        if $p^k > p^{k-1}$ **then**
            inject $p^k - p^{k-1}$ new individuals into P;
            decode $p^k - p^{k-1}$ new individuals into P;
            sort current population P by fitness value;
        else
            remove $p^{k-1} - p^k$ individuals from P with the least fitness values;
    split population P in elite and non-elite sets;
    copy elite individuals to next generation $k + 1$;
    insert mutants to P;
    perform mating to remaining population;
    $B \leftarrow$ best solution of population P;
    if $B > B^*$ **then**
        update current best solution $B^* \leftarrow B$;
    update generation counter $k \leftarrow k + 1$;
return B^*;

3.2 BRKGA-Race

The second approach that we explored was to introduce the principles adopted by irace [2] in BRKGA, designing a metaheuristic that tunes itself while solving the optimization problem.

The best configuration for a metaheuristic is defined as the parameter setting (i.e. the values for each parameter) that leads to the best possible empirical performance on a set of problem instances [7]. As described in detail in Section 2.2.1.1, irace seeks to find the best configuration by executing mainly three steps: (1) sampling new configurations according to a particular probability distribution, (2) selecting the best configuration by means of racing, and (3) updating the sampling distribution biasing them toward the best configurations. These steps were the foundation of our proposal.

Regarding steps (1) and (3), our proposal follows the irace methodology with minor modifications, that will be explored ahead. Step (2), on the other hand, presents a significant difference to irace. Irace selects the best configurations by the racing procedure. In each step of the race, multiple candidate configurations are evaluated on a single instance. When the configurations were evaluated on a minimum set of instances, they are compared using a statistical test. The test evaluates a series of independent samples of each configuration ran on different instances and compares these series pair-by-pair to eliminate those with the worst performance.

When aiming to tune the metaheuristic in an *online* manner, there is not a set of instances available to test different configurations. There is only the instance at hand that needs to be solved by the algorithm. Our goal is not only to find the best configuration for the metaheuristic but also to obtain a good solution to the problem instance at hand in a reasonable amount of time. By doing so, we can save resources by eliminating the need to tune the parameters before the optimization process. Besides, online tuning can provide an appropriate balance between diversification and intensification leading to a more efficient search process and better solutions [8].

Without having an instance set to evaluate the configurations, we propose to adapt the procedure by adopting multiple populations. The race is performed by running different configurations in different independent populations. The configurations are evaluated against each other at the end of each race. The best configurations lead to surviving populations that attract individuals from non-surviving populations. This way, we are able to test different configurations while solving the problem at hand. We also benefit from the advantages of having multiple populations, which allows us to explore the solution space at different locations and exchange valuable information from one population to another.

The concept of the proposal is illustrated in Figure 3.1. The first step is an algorithm setup in which we define how many races will be executed and some settings of the evaluations. Then, we start a race. We need to do a setup, defining the time budget for the race and how many configurations will be evaluated within it. If it is the first race, we generate new configurations and run them. The configurations are compared against each other and the surviving populations are selected. The best individuals from the dying populations migrate to the surviving ones. Then, we can start a new race. In the following races, we run the surviving populations again, giving them more time to improve. After running the surviving populations we repeat the process of generating new configurations, evaluating them, selecting the surviving populations, and migrating individuals until the stopping criteria are met.

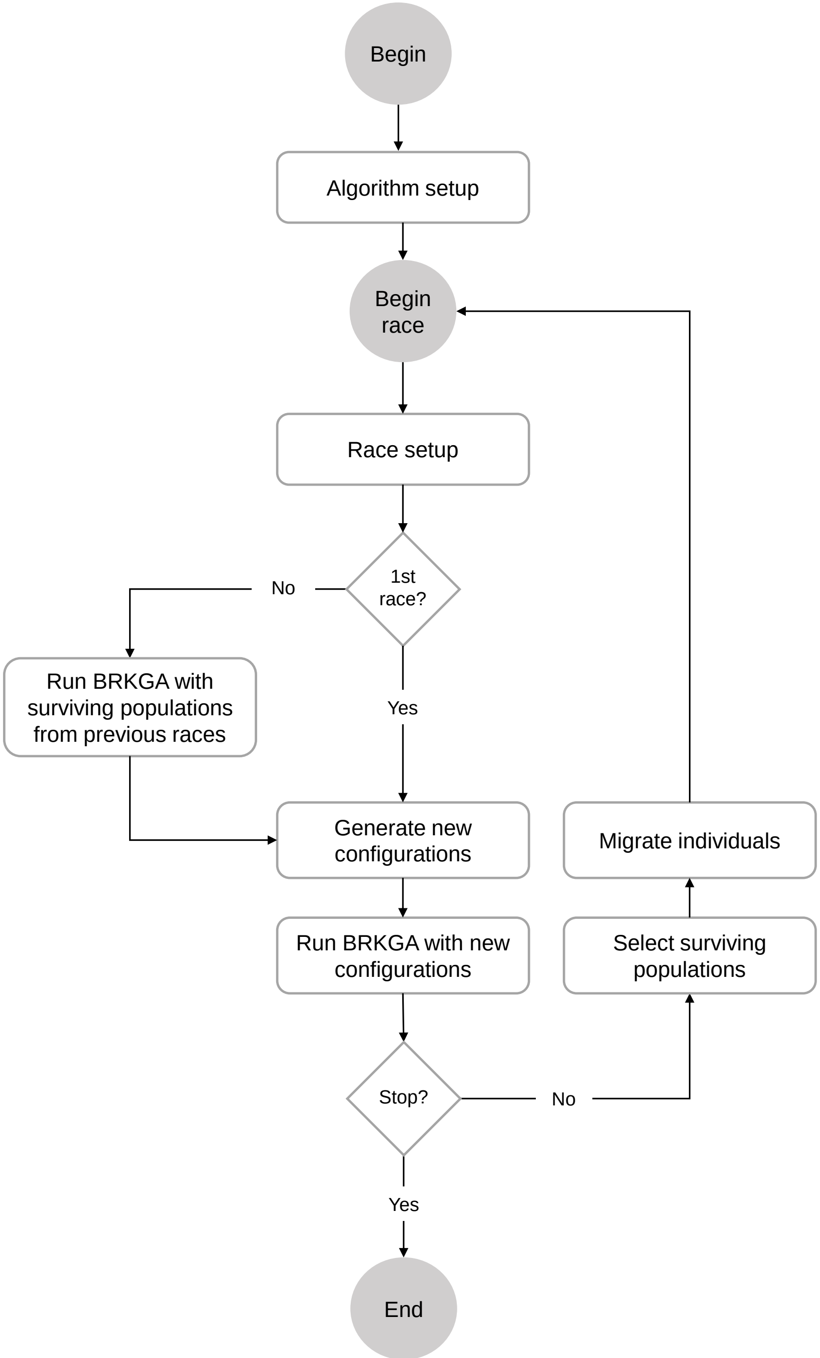

Figure 3.1: Flowchart of the BRKGA-Race method.

3.2.1 Algorithm and race setup

The user needs to provide three pieces of information to the algorithm. First is a maximum time limit $maxTime$. This is how much time the user is willing to spend in the optimization process. We consider a time limit, not a number of iterations. The time taken by the algorithm to execute one iteration varies according to the problem's size and complexity and the decoder implementation. Defining a time limit grants users more predictability and control over the process. The second user input is the number of parameters being tuned, and the third are the lower and upper bounds of each parameter.

Following the methodology adopted to the Random Parameter Values method, described in Section 3.1, we also tuned the four main BRKGA parameters – the population size (p), the proportion of elite individuals (p_e), the proportion of mutant individuals (p_m), and the probability of inheriting a gene from parents from the elite set (ρ_e). We provided the same intervals (upper and lower bounds) suggested by Gonçalves and Resende [1].

As seen in Figure 3.1, in the first step the algorithm setups the number of races it will execute based on the number of parameters being tuned. To define the number of races, we adopt the same equation provided by López-Ibáñez et al. [2]. The number of races assumes a minimum value of two and it is a function of the number of parameters (n_p) being tuned, as it can be seen in Equation (3-1). In our case, since we are tuning four parameters, we execute four races.

$$n = 2 + \log_2 n_p \tag{3-1}$$

In a race, γ_i populations evolve for a determined b_i amount of time. Each population has a configuration associated to it. Each race has a time budget to evaluate its configurations, calculated as a function of $maxTime$ and n. The time budget b_i for each race i ($i = 1, 2, ..., n$) is defined by Equation (3-2). This equation follows the budget equation of irace [2].

$$b_i = (maxTime - b_{used})/(n - i + 1) \tag{3-2}$$

With Equations (3-1) and (3-2), we aim to define how many races the algorithm will execute and how much time it will spend on each race. Once the time budget b_i for a race is defined, we need to set the number of configurations that will be evaluated in each race. To do so, we set a minimum time of 180 seconds for each configuration, with an increment of a fixed percentage k (a value within the interval $[0, 1]$) on the following races. The minimum time t_{min} is then calculated as described in Equation (3-3), where i is the number of the current race.

$$t_{min} = 180 \cdot (1 + k)^{(i-1)} \tag{3-3}$$

The number of configurations γ_i to be evaluated in each race – that is, the number of populations that will be run – is defined by the time budget b_i divided by the minimum time t_{min}. With this approach, in the first race, more configurations are evaluated for less time, providing greater diversification. By the end, fewer configurations are evaluated for longer aiming to intensify good solutions.

3.2.2
The first race

Having the time budget b_i, the time to evaluate each configuration t_{min}, and the number of configurations that will be evaluated in hands, we can move forward into generating new configurations and running BRKGA to obtain solutions while evaluating the provided settings.

In the first race, the initial set of configurations is generated by uniformly sampling the user-provided parameter space, as it is done in Irace [2]. Then, we apply each of these configurations to BRKGA and run it for t_{min} seconds.

3.2.3
Population selection and individuals' migration

Once we have run BRKGA with all generated γ_i configurations, we evaluate them against each other to select the populations that will survive to the next race. Irace [2] evaluates multiple candidate configurations on multiple instances. After evaluating the candidate configurations in a few instances, a statistical test is performed on the results. If some configuration is observed to be performing worse than at least another configuration, it is removed from the race. By default, Irace [2] uses the non-parametric Friedman's two-way analysis of variance by ranks (the Friedman test [52]) to select which configurations will be discarded during the race.

In our case, we have multiple configurations being run on the same instance. If we were going to perform the tests like Irace [2], we would need to run the same configuration on one instance a few times varying the seed value. This would take a long time, and we would need to run the same configuration over again, even if it was not promising. Since our goal is not only finding a good configuration but also finding the solution to the problem instance at hand, we opted for a different approach.

We seek to identify which population is doing best by observing the improvement information of each experiment, which consists of γ time-series describing the improvement profile of the population or the current best solution for every iteration of BRKGA. The best population will survive to the next step. In this case, we cannot compare the time series using the Friedman test, because the test is best suited for independent data points and our improvement series is composed of dependent data. We propose to compare the pairs of time series for state-wise stochastic dominance.

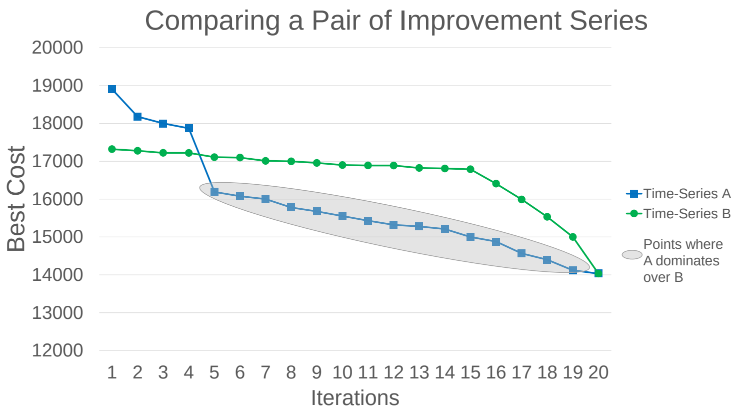

Figure 3.2: Comparing a pair of improvement series and evaluating the dominance of A over B.

Table 3.1: Two series of solution values throughout the algorithm's execution with different lengths.

	Iterations																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	189	189	189	189	172	172	172	172	172	161	161	161	161	161	161	161	145	145	145	145
B	173	172	172	172	172	172	169	169	169	169	140	140	140	140	140					

Stochastic dominance [64] is a partial ranking measure between random variables applied in game theory and economics. Given a pair of *gambles*, we can determine which gamble dominates over the other. Random variable A is state-wise dominant over random variable B if A gives at least as good a result in every state (every possible set of outcomes) and a strictly better result in at least one state.

We calculate a measure of dominance for each pair of time series considering the proportion of points in time-series A that are strictly better than the points in time-series B. Consider Figure 3.2. Population A, represented by time-series A constituted by improvement points, is 75% dominant over population B since it has 15 points that are strictly better (lower-valued) within a total of 20 points.

Regarding the population size and the t_{min} time available to evaluate each configuration of γ, one population may be run for a different number of iterations than the others, generating improvement series of different lengths. When this is the case - when we have different-sized series to compare - we standardize their length by proportionally removing points to match the shorter vector. See Table 3.1. Let's say we have configuration A, which ran for 20 iterations, and configuration B, which ran for 15 iterations. Since we record the best cost in every iteration, there is usually repetition.

We take the longer series, in this case, Series A, and count the frequency of each value, as can be seen in Table 3.2. Then, we produce a new vector by applying the same proportion of each value in Series A, but with Series B's length. If some adjustment is needed regarding rounded values, we truncate the vector to match the size exactly. With this approach, we can preserve the series' distribution and perform the dominance evaluation as previously described.

Table 3.2: Matching size of Series A with Series B.

Solution Value	**Frequency**	**New Frequency**
189	4	3
172	5	4
161	7	6
145	4	2
Vector's length	**20**	**15**

After all configurations γ_i were evaluated pair-by-pair and we have a measure of dominance for each combination of series (i.e. A over B, B over A), we calculate the average dominance for each series. We define the γ_i^s populations to survive to the next race, by selecting a percentage p^s of the top configurations based on the average dominance values. The populations to survive is then defined by $\gamma_i^s = min(1, ceil(p^s * \gamma_i))$. We also rank the populations using the dominance measure. The ranking is used to bias the parameter distribution as will be described in the next subsection.

In order not to waste useful information about the populations that did not survive, we perform migration. Migration occurs by selecting the best individual (best chromosome) from each population that will not survive to the next race and inserting them into the surviving ones. To preserve the configuration of the surviving populations, we remove the worse $\gamma_i - \gamma_i^s$ chromosomes to maintain the population size. The processes of selection and migration described are briefly illustrated in Figure 3.3.

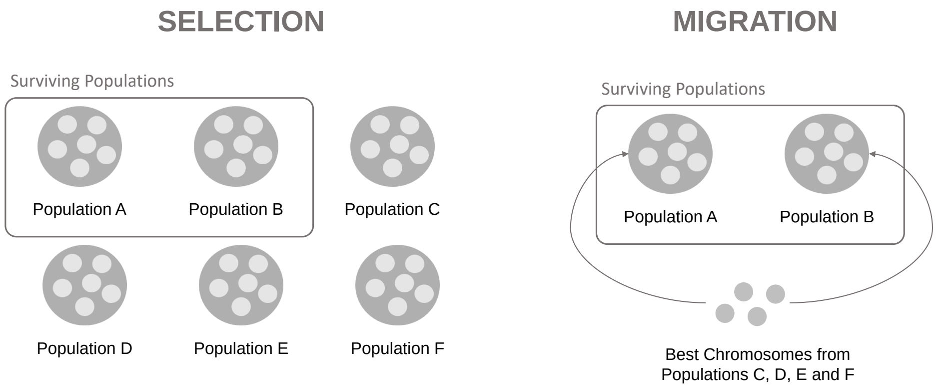

Figure 3.3: Illustrations of Selection and Migration procedures. On the left, we can see the selection of 2 populations (A and B) in a set of 6 populations that were evaluated after a race. On the right, we see an illustration of the best chromosomes from the non-surviving populations (C, D, E, and F) being migrated into the surviving ones.

3.2.4
Following races

After the first race is completed, a new race starts with a few modifications. In the race setup, the time budget b_i is updated using Equation (3-2) and the time to evaluate each configuration t_{min} is updated by an increase of 25%. Considering the number of surviving populations from the previous race that will continue evolving in this race, we calculate how many new configurations will be introduced. The number of configurations to be evaluated in each race is defined by the time budget b_i divided by the minimum time t_{min}, as stated earlier. The number of new configurations is given by $\gamma_i^n = (b_i/t_{min}) - \gamma_i^s$.

We initially run the surviving populations from the previous race for t_{min} seconds, providing it more evolution time. To generate the γ_i^n new configurations we apply the idea to adapt the parameters probability distribution proposed by López-Ibáñez et al. [2]. At the end of each race, we produced a set of configurations γ_i that are ranked according to the dominance measure of each population. To generate a new configuration, a parent configuration is sampled from the γ set with proportional probability to the dominance rank. This means that higher-ranked configurations have a higher probability of being selected as parent configurations.

We sample a new value for each parameter X_d. If X_d is defined within the interval $[\underline{x_d}, \overline{x_d}]$ we sample this value from the normal distribution $N(x_d^i, (\sigma_d^i)^2)$. The distribution's mean x_d^i is the value of parameter X_d in the parent distribution. The standard deviation σ_d^i is initially defined as Equation (3-4), as seen in [2].

$$\sigma_d^i = \frac{\underline{x_d} - \overline{x_d}}{2} \tag{3-4}$$

The standard deviation σ_d^i decreases at each race according to Equation (3-5), where n is the number of populations and n_p the number of parameters being tuned. The idea is to lead sampled values closer to the parent configuration.

$$\sigma_d^i = \sigma_d^{i-1} \cdot (\frac{1}{n-1})^{1/n_p} \tag{3-5}$$

After the new configuration is generated, we run BRKGA applying this configuration and repeat these steps until we have evaluated the established γ_i^n new configurations. Having the improvement series obtained from both old configurations and new ones, we perform the selection and migration steps, concluding the race. These steps are repeated until we reach the number of races defined by Equation (3-1).

While running the races and evaluating the generated configurations, we record the best solution cost obtained for each run. By running the surviving populations for longer, we are able to continue searching for solutions in promising spaces. With the proposed approach we can generate and test different configurations while solving the problem.

4
Experiments and Discussion

In the experiments, our main goal was to evaluate the proposed parameter control methods against the state-of-the-art tuning method, Iterated Racing (Irace) [2]. We initially present the benchmark problems that were selected in order to represent pertinent classes of combinatorial problems and how the use case (our benchmark dataset) was generated. Then, we present the methodology that we used to conduct our experiments, both for the random parameters approach and for the BRKGA-Race. Alongside the experiment methodology, we present and discuss the obtained results.

4.1
Benchmark Problems

We evaluated the proposed methodology in three classic problems of the literature. By choosing these problems we aimed to represent three relevant classes of combinatorial problems: scheduling, routing, and location. The chosen problems to represent each one of these classes are the flowshop scheduling problem with flow time minimization, the traveling salesman problem, and the set covering problem. In this section, we present the problem's description and the instances used. We also describe the decoders and local searches that were implemented to tackle these problems.

Since the output solution generated by BRKGA is not necessarily optimal, local search heuristics are usually employed to improve them. The hybridization of evolutionary algorithms with local search procedures, in order to improve the solutions, was first proposed in 1989 by Moscato [65]. These procedures work by modifying solutions and trying to escape from local optima. The adoption of local searches with BRKGA can be seen in Londe et al. [13], Andrade et al. [3], Pinto et al. [15], Mauri et al. [12], and Biajoli et al. [29], just to name a few.

Including a local search in the decoding process can lead to improved solutions but can also lead to increased decoding time and complexity. A local search includes additional steps in this process and may add running time variability. In this work, we performed our experiments considering the decoders with and without local searches, as seen in Londe et al. [13] and Andrade et al. [3], to observe the differences between both approaches.

We opted for first improvement local searches to obtain solution improvements faster. In the first improvement approach, a modification is applied to the solution as soon as it leads to an improvement in the solution value. While in the best improvement approach, all possible moves are evaluated and only the one that brings the best improvement is performed. It is expected that, on average, first improvement local searches to have faster convergence rates. In the BRKGA-Race method, decoding time is crucial since we evaluate multiple configurations sequentially – each one with limited time. The details of the implemented local search heuristics are available within the decoder's description for each problem.

4.1.1 Flowshop Scheduling Problem

A Flowshop Scheduling Problem (FSP) consists of a set of n jobs that need to be processed at a certain sequence in m machines [66]. Each job can be processed on only one machine at a time and the jobs cannot be split. Equivalently, each machine can process only one job at a time. There are some variations in the objective function for this problem. In this work, we adopt the minimization of the flow time. The flow time is given by the sum of the completion time of each job on the last machine.

The solution encoding for the FSP is composed of a random-key – a random value in the range of real numbers [0, 1] – for each job. Thus, the chromosome presents n genes, each position linked to one specific job. The decoding process is illustrated in Figure 4.1. To decode the chromosome into a feasible solution, the decoder sorts the random-keys and the obtained sequence corresponds to the jobs processing order [3]. Every solution obtained with this permutation decoder is feasible – a valid solution to the problem. Note that the decoders always take $n \log_n$ operations to be carried out since it is a classical sorting algorithm.

The first improvement local search included in the FSP decoder consists of selecting one job and trying to switch positions with the following jobs [67]. The local search conducts a movement when it leads to an improved solution. After a movement that led to an improvement is performed, the search starts again from the beginning. It stops when it cannot make any more improvements.

To evaluate the proposed methodology, we used all 120 benchmark instances of the FSP proposed by Taillard [68]. The smallest ones contain 20 jobs and 5 machines, and the larger ones contain 500 jobs and 20 machines. More information on the instance's dimensions are available in Appendix A.1.

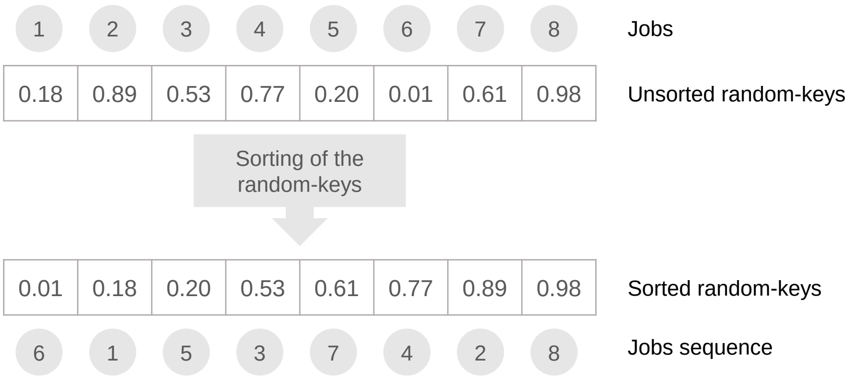

Figure 4.1: Decoding of a chromosome into a feasible solution of the FSP. Adapted from [3].

4.1.2 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) consists in finding the shortest route for a traveling salesman who starts in one city and must visit every city on a given list and then return to the origin. The cost of traveling from any city i to any other city j is known, and we aim to obtain the tour with the least possible cost to visit every city. A TSP instance is given by a complete graph G on a node set $V = 1, 2, ..., n$, where n is an integer, and by a non-negative cost function that assigns a cost $c_{i,j}$ to the arc (i, j) for any $i, j \epsilon V$ [69].

To encode a solution for the TSP, we set a random-key for each city. The chromosome is assembled by n genes, each position linked to one city. This decoding procedure is implemented in the BRKGA-MP-IPR package [17]. The decoding process is illustrated in Figure 4.2. It works like the FSP decoder. In order to decode the chromosome into a feasible solution, the decoder sorts the random-keys and the obtained sequence corresponds to a tour to be performed by the salesman. As in the FSP, every solution obtained with this permutation decoder is a valid solution to the problem and the decoder always executes $n \log_n$ operations.

The local search implemented for the TSP is the 2-OPT algorithm, proposed in [70]. In this local search, we remove two arcs from the route, reconnect their vertices in two new arcs and calculate the new tour distance. If the swap leads to a shorter travel distance, we update the current route. This procedure is repeated until no more improvements are found.

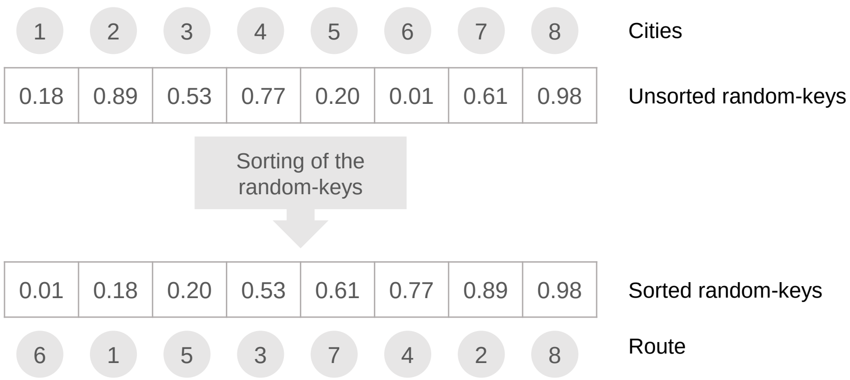

Figure 4.2: Decoding of a chromosome into a feasible solution of the TSP.

To evaluate our methods on the TSP, we used all 103 instances available on the TSPLIB [71]. These instances contain examples with sizes from 14 cities up to 85,900 cities.

4.1.3 Set Covering Problem

The Set Covering Problem (SCP) consists of finding a minimum cost coverage for a set of objects. Let $I = 1, 2, \ldots, m$ be a set of objects. A collection of subsets $P_1, P_2, \ldots, P_n$ covers the objects of I and a cost c_j associated with each subset $P_j, j = 1, 2, \ldots, n$. The goal is to find the minimum cost coverage so that each object is covered by at least a subset [72, 73].

A SCP solution is encoded by a random-key vector x that contains n keys in the range of real numbers [0, 1]. The j^{th} key corresponds to the j^{th} subset of A, where A is the collection of subsets. Figure 4.3 illustrates the first part of the decoding process. The decoder selects the subset j to be in J^* if $x_j \geq 0.5$.

Suppose after the first part of the decoding process the resulting set J^* is feasible. In that case, the fitness value of the solution is calculated by computing the cost of the subsets included in the solution. On the other hand, if J^* is unfeasible, a greedy algorithm [74] includes subsets iteratively until the solution becomes feasible. It selects the subset with the smallest ratio between the cost of a subset and its cardinality (the number of objects covered by it). Every time a subset is included in the solution, the cardinality of the remaining subsets is recalculated removing covered objects.

The SCP decoder is more complex than the previous ones. It has to seek feasible solutions in an iterative process, which can take from zero to n

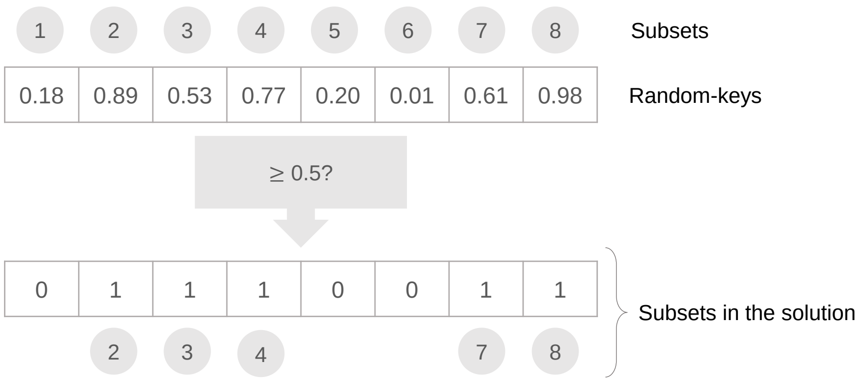

Figure 4.3: Decoding of a chromosome into a preliminary solution of the SCP.

operations. It eventually includes too many subsets in the solution to guarantee full coverage. In this case, we opted to implement a local search that would simplify the obtained solution by trying to remove redundant subsets. The local search iterates over each set in the feasible solution and try to remove it. If the solution remains feasible after removing one set, it completes the removal and proceeds to the following subset. This removal of superfluous elements is observed in the Set Cover heuristics proposed by Feo and Resende [75].

We considered 70 instances of the Set Covering Problem available in the OR-Library [76]. Where 50 of these instances are test-problem sets 4 to 6 and A to E from Beasley [72] and 20 are the test-problem sets E to H from Beasley [73] (NRE, NRF, NRG, and NRH sets). The description of the instance's dimensions is available in Appendix A.2. For more information on the files, please refer to the OR-Library website [77].

4.2 Computational Environment

The experiments were conducted on a cluster of identical machines with Intel Xeon E5-2650 processors, CPU with 2.0 GHZ (12 cores / 24 threads), and 128 GB of RAM running CentOS Linux 6.9. Times are reported in real-time seconds, excluding the necessary time for instance loading and algorithm's warmup. We relied on the BRKGA-MP-IPR framework proposed by Andrade et al. [17] as the foundation to execute BRKGA in these experiments. All algorithms were implemented in the language Julia [78] version 1.6.

4.3 Setting the Use Case

In this work, we aim to evaluate BRKGA adaptations that are embedded with parameter control approaches seeking to understand if they can lead to good results in solving problems while auto-tuning the algorithm. To evaluate the results' quality, we compare them with BRKGA tuned by the state-of-the-art approach to parameter tuning, Irace [2].

To set up the benchmark for our experiments, we tuned the three classical problems presented in Section 4.1 with Irace. We also considered two variants for each problem - the "pure" decoder, represented as the "NLS" approach ("no local search") and the decoder associated with a local search, denoted as the "LS" approach. It is needed to set some configurations for Irace's execution, which we left with default settings. Irace was set with a budget of 2,000 experiments and 50 distributed machines for parallel execution. For the instance training set, we selected a representative sample with 20% of the problem instances available. We set the intervals (lower and upper bound values) for each parameter as recommended by Gonçalves and Resende [1], available on Table 2.2. Table 4.1 presents the elapsed time to tune the problems with its variations.

Table 4.1: Elapsed time (in hours and days) to tune each problem. A decoder of type "NLS" indicates the pure decoder, without local search. While "LS" indicates the version with local search included. (*) Indicates that it was not possible to terminate Irace, after several days of execution.

Problem	**Decoder**	**Time (Hours)**	**Time (Days)**
Flowshop Scheduling Problem	NLS	88.80	3.70
	LS	638.15	26.60
Traveling Salesman Problem	NLS	279.78	11.70
	LS	*	*
Set Cover Problem	NLS	131.63	5.50
	LS	+15.35	+0.64

It is possible to observe in Table 4.1 that the tuning process is very time-consuming, taking at least three days to perform a full assessment. In the worst case, it took almost 27 days to terminate. It was not possible to obtain a configuration for the TSP with the local search (TSP LS) version. The tuning algorithm ran for several days and couldn't end the process. After some crashes on the server, we terminated the execution. In this particular case, we used the configuration provided for the TSP without local search to set the TSP LS version.

The tuning process was also interrupted for the SCP with the local search. Only one out of four iterations was completed and the time displayed in Table 4.1 refers to the duration of this one iteration. The parameter values used for the tuned version of the SCP LS are therefore the best configuration found on the first iteration of Irace.

After tuning the problems, we set up BRKGA with the configuration provided by Irace [2]. Then we ran BRKGA for all problems and all problem instances, considering the decoders with and without local search, for 30 independent runs with a time limit of 3,600 seconds (one hour) each. The results obtained from these runs were set as our benchmark for this study.

4.4 Random Parameter Values

Our investigation started seeking to answer the question: "Can adopting random parameter values in BRKGA lead to results as good as the results obtained by the algorithm tuned with Irace?". After adapting BRKGA to be able to generate and adopt random parameter values at each generation, as described in Section 3.1, we set the parameter intervals as recommended by Gonçalves and Resende [1], which were the same values that were used in the tuning procedure that is described in the use case setup (Section 4.3).

We ran the random parameter valued version of BRKGA for all three problems, all problem instances, and 30 independent runs with a time limit of 3,600 seconds considering the decoders without the local search. The results of the random parameter valued are labeled as BRKGA-Random NLS, and the results from the use case, resultant of BRKGA tuned with Irace are labeled as BRKGA-Tuned NLS, where "NLS" stands for "No Local Search" included in the decoder. The metrics we used to evaluate the results of the experiment were the following:

- $ADev_B$: The average relative percentage deviations from the best solution known in the literature for that problem instance (most of the times, the optimal solution);
- $MDev_B$: The median relative percentage deviations from the best solution known in the literature;
- $ADev_T$: The average relative percentage deviations from the best solution obtained by BRKGA-Tuned NLS, when setting the use case;

Table 4.2: Aggregated results of BRKGA-Random NLS compared to BRKGA-Tuned NLS for the three studied problems.

Problem	BRKGA TUNED NLS		BRKGA RANDOM NLS		RANDOM-TUNED RELATIVE	
	$ADev_B$	$MDev_B$	$ADev_B$	$MDev_B$	$ADev_T$	$MDev_T$
FSP	7.18	6.09	3.28	3.45	-3.57	-3.12
TSP	1934.32	908.77	1607.68	1247.36	-46.71	-43.77
SCP	28429.59	1268.76	123502.93	17135.85	518.42	458.46
All problems	8410.50	162.22	40492.12	7.11	147.38	-2.82

– $MDev_T$: The median relative percentage deviations from the best solution obtained by BRKGA-Tuned NLS, when setting the use case.

In summary, the metrics with the "B" subscore are regarding the distance to the best solution found in literature, while the "T" subscore regards the solutions obtained by the benchmark assembled for this study. All of the metrics above are calculated identically, only changing the reference value. They are computed by $((V - V_{ref})/V_{ref}) \cdot 100$, where V is the value of the best solution obtained by the current algorithm among 30 independent runs, and V_{ref} is the best solution value in the adopted reference.

The aggregated results are displayed in Table 4.2. The results indicate that for the FSP and TSP the results obtained by BRKGA-Random NLS are better (lower) than the results obtained by BRKGA-Tuned NLS, which can be observed in the $ADev_T$ and $MDev_T$ metrics that display negative values for these two problems. When looking at the $MDev_T$, the solution's values are around 3.1% lower for the FSP than the solutions obtained by the tuned version of BRKGA. For the TSP, the solution's values are around 40% lower. However, for the SCP, the results obtained with the random parameter valued version are 4.5x worse than the solutions provided by BRKGA-Tuned, according to the median metric. Later, we discuss such result.

Exploring the Flowshop Scheduling Problem, Table 4.3 presents the results per instance group, in which each group contains 10 instances. Observe that BRKGA-Random NLS presented superior results than BRKGA-Tuned NLS for every instance group. For the biggest instance group (500 x 20), BRKGA-Random NLS presented results around 6% lower.

Figure 4.4 presents the distribution of the deviations regarding the best solution found in literature for both BRKGA-Random NLS and BRKGA-Tuned NLS. It is possible to observe in the histogram that the deviations of the BRKGA-Random NLS are concentrated in the first half of the image, indicating lower values and lower variability.

Table 4.3: Results of BRKGA-Random NLS compared to BRKGA-Tuned NLS for the Flowshop Scheduling Problem.

Problem	BRKGA TUNED NLS		BRKGA RANDOM NLS		RANDOM-TUNED RELATIVE	
	$ADev_B$	$MDev_B$	$ADev_B$	$MDev_B$	$ADev_T$	$MDev_T$
20 x 5	3.71	3.51	0.70	0.55	-2.89	-2.64
20 x 10	3.62	3.60	0.77	0.64	-2.73	-2.69
20 x 20	2.65	2.64	0.65	0.52	-1.94	-1.93
50 x 5	4.95	4.85	2.70	2.62	-2.14	-2.10
50 x 10	6.12	5.94	3.45	3.39	-2.51	-2.46
50 x 20	5.47	5.36	3.19	3.13	-2.15	-2.15
100 x 5	6.72	6.81	3.28	3.25	-3.21	-3.44
100 x 10	7.89	7.74	4.10	4.09	-3.49	-3.46
100 x 20	7.33	7.04	4.14	4.13	-2.97	-2.74
200 x 10	11.47	11.44	4.73	4.72	-6.04	-6.07
200 x 20	11.73	11.78	5.04	5.02	-5.97	-6.04
500 x 20	14.42	14.41	6.64	6.62	-6.80	-6.84
Full set	7.18	6.09	3.28	3.45	-3.57	-3.12

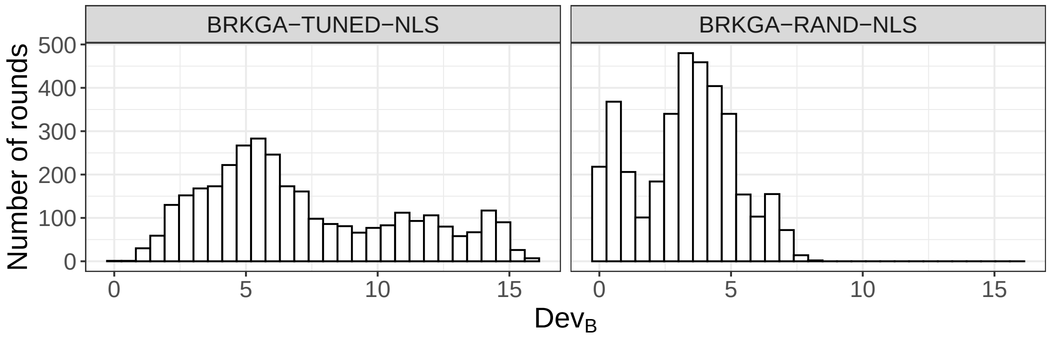

Figure 4.4: Distribution of relative percentage deviations from the best-known solution for the FSP instances.

Regarding the Traveling Salesman Problem, the results that are shown in Table 4.4 exhibit that BRKGA-Random NLS presented better results on every instance group. The groups were divided considering the size of the instances (which consists of the number of cities). The instance size did not appear to have a great influence on the performance of BRKGA-Random NLS.

Figure 4.5 shows the distribution of the relative percentage deviations (RPD) from the best known solutions for the TSP instances. While both methods' data is concentrated on the left portion of the graphs, BRKGA-Random NLS shows a greater number of solutions closer to the best-known solutions, and BRKGA-Tuned NLS presents some outliers with noticeable deviations.

The results of the Set Covering Problem are not like those observed for

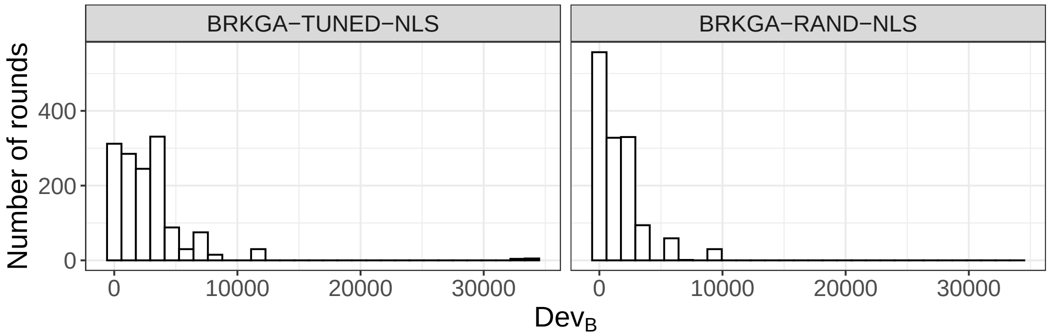

Figure 4.5: Distribution of relative percentage deviations from the best-known solution for the TSP instances.

the previous two problems. Table 4.5 summarizes the results for each instance group. Instances were grouped according to the problem sets available on the OR Library [77]. Results show that BRKGA-Random NLS performed poorly than BRKGA-Tuned NLS in almost every group, except one (“E” instance group). This instance group has small absolute values when compared to other instance groups. For example, optimal solution values of the “E” instance group range are all equal to 5, while in other groups such as group “4” instance group the instances' absolute values range from 429 to 641. Thus, relative percentage deviation metrics are much more sensitive on the “E” instance group, which can lead to a distortion in the final results. On average, solutions obtained with BRKGA-Random NLS were around 5x worse than the ones obtained with BRKGA-Tuned NLS. Both results, however, are far from the best-known solutions and cannot be considered good quality results.

Observing the distribution of the RPD values of the Set Covering

Table 4.4: Results of BRKGA-Random NLS compared to BRKGA-Tuned NLS for the Traveling Salesman Problem.

Problem	BRKGA TUNED NLS		BRKGA RANDOM NLS		RANDOM-TUNED RELATIVE	
	$ADev_B$	$MDev_B$	$ADev_B$	$MDev_B$	$ADev_T$	$MDev_T$
14 - 48	90.97	71.18	6.94	5.40	-37.81	-38.69
49 - 100	377.04	417.09	39.79	43.10	-69.56	-71.64
101 - 130	523.48	504.93	67.76	61.72	-72.05	-71.76
131 - 198	3797.26	679.90	275.12	122.26	-73.35	-72.83
199 - 318	1022.98	949.62	255.82	239.98	-67.86	-67.43
319 - 575	1402.15	1377.31	527.15	524.62	-56.37	-57.29
576 - 1291	2471.78	2621.72	1237.09	1256.74	-46.70	-47.50
1292 - 2152	3849.32	3396.60	2338.46	2201.96	-37.78	-36.99
2153 - 5934	5777.40	4310.10	4627.93	3510.73	-21.12	-20.86
Full set	2815.54	2250.51	1607.68	1247.36	-46.71	-43.77

Problem (Figure 4.6), we can notice significant dispersion in both methods. This is mainly because the solution value's magnitude differs from one instance group to another, leading to significant variability when calculating RPD values. The figure points to a greater volume of data closer to the best-known solutions in the BRKGA-Tuned NLS chart.

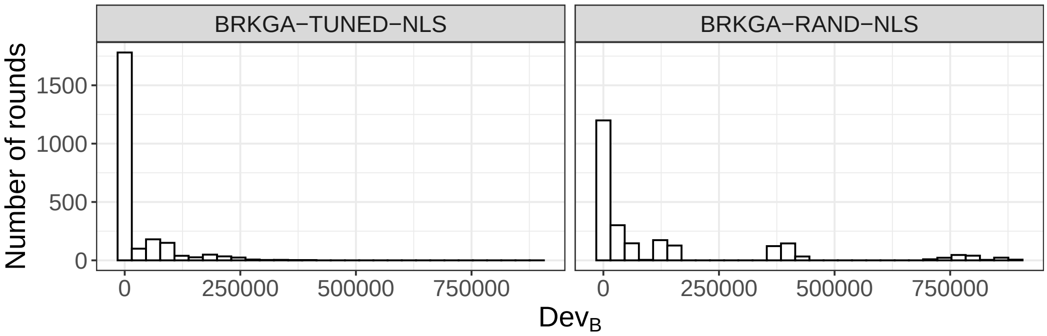

Figure 4.6: Distribution of relative percentage deviations from the best-known solution for the SCP instances.

As stated earlier, the results observed for the FSP and TSP differ significantly from the SCP. One of the main differences between how these problems are processed can be observed in their decoders. The decoders for the FSP and the TSP are the same. They both consist of permutation steps, and always perform the same number of operations. The decoder of the SCP is different. When it cannot find a feasible solution, it iterates trying to include another subset in the solution. This lack of feasibility activates a greedy

Table 4.5: Results of BRKGA-Random on SCP by instance group.

Problem	BRKGA TUNED NLS		BRKGA RAND NLS		RANDOM-TUNED RELATIVE	
	$ADev_B$	$MDev_B$	$ADev_B$	$MDev_B$	$ADev_T$	$MDev_T$
4	33.02	31.45	257.03	227.60	170.63	146.38
5	791.84	769.33	4759.29	4520.64	461.12	435.52
6	106.35	94.45	880.89	614.73	387.84	254.92
a	1633.04	1637.57	20264.88	20120.65	1093.07	1068.15
b	5456.16	5372.30	65757.92	65574.49	1105.77	1102.90
c	3576.63	3522.75	36394.05	36581.28	909.47	894.43
d	12704.92	12410.58	128062.58	130607.95	918.28	920.09
e	234.13	220.00	24.00	20.00	-59.81	-62.50
nre	47947.01	45650.00	386804.83	387839.29	728.57	740.30
nrf	98421.79	96800.00	787512.58	782696.43	710.79	709.01
nrg	71846.59	69424.49	142682.53	142175.30	103.68	106.90
nrh	211241.51	199630.72	397179.69	399741.53	96.73	103.74
Full Set	28429.59	1268.76	123502.93	17135.85	518.42	458.46

Table 4.6: Aggregated results of BRKGA-Random LS compared to BRKGA-Tuned LS for the three studied problems.

Problem	BRKGA TUNED LS		BRKGA RAND LS		RANDOM-TUNED RELATIVE	
	$ADev_B$	$MDev_B$	$ADev_B$	$MDev_B$	$ADev_T$	$MDev_T$
FSP	2.87	2.60	4.60	4.60	1.59	1.37
TSP	572.55	3.36	501.15	3.56	-10.02	-0.03
SCP	29.30	20.93	28.71	22.22	-0.15	0.00
All problems	204.81	4.56	195.82	6.08	-3.39	0.00

procedure within the decoder.

To access differences in BRKGA-Random performance, we included local searches to all decoders. After including the local searches we ran BRKGA-Random LS for all problems and all problem instances, for 30 independent runs and a time limit of 3,600 seconds for each run. This version is labeled as BRKGA-Random LS. Table 4.6 summarizes the obtained results.

Including a local search within each decoder led to overall better results in all methods. When comparing the BRKGA-Random LS and BRKGA-Tuned LS versions, we can observe better results for the tuned version in the FSP by 1.6%, according to the average metric. The BRKGA-Random LS, however, outperformed the tuned version in TSP (by 10%) and SCP (by 0.15%). In general, the results of the random version with local search included were very competitive when compared to the tuned algorithm.

When looking at the FSP and comparing the results of BRKGA-Random NLS and LS, displayed in Tables 4.3 and 4.7, we can see that the results obtained by the algorithm without local search (NLS) were better than the ones obtained by the algorithm with local search – see metrics $ADev_B$ and $MDev_B$. In the FSP, it was more valuable to decode solutions faster and run a greater number of iterations than to improve locally solutions.

Since the results for the BRKGA-Random NLS were better, we may suppose that the increase of the complexity in the decoder by adding the local search procedure, had an impact on the method's overall performance. The solution values of the tuned version presented an increase in the solution quality of 68% by including the local search procedure. On the random version, adding the LS led to a decrease in the solution quality of 40%. Comparing all four methods approached so far, BRKGA-Tuned LS led to the best results, followed by BRKGA-Random NLS, BRKGA-Random LS, and BRKGA-Tuned NLS as seen in Table 4.8.

When observing the distribution of the deviations on the BRKGA-

Random LS on Figure 4.7, we can also notice a great increase in dispersion of the values with the new decoder with local search. A decrease in dispersion was observed in the BRKGA-Tuned LS.

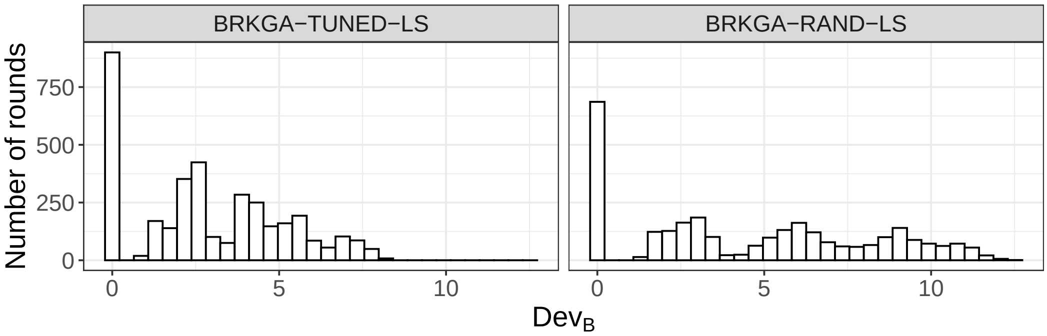

Figure 4.7: Distribution of relative percentage deviations from the best-known solution for the FSP instances (with Local Search).

In Table 4.9 we can observe the results for the TSP, considering the decoder with the local search. In this case, we can also notice the decrease in solution quality, when comparing the random version to the tuned version with and without local search. When dealing with the decoder without local search, the random version led to results around 46% better than the tuned version. When including the local search within the decoder of the TSP, BRKGA-Random LS solution values are, on average, still 10% better than BRKGA-Tuned LS.

Table 4.7: Results of BRKGA-Random on FSP by instance group.

Problem	BRKGA TUNED LS		BRKGA RAND LS		RANDOM-TUNED RELATIVE	
	$ADev_B$	$MDev_B$	$ADev_B$	$MDev_B$	$ADev_T$	$MDev_T$
20x5	0.00	0.00	0.00	0.00	0.00	0.00
20x10	0.00	0.00	0.00	0.00	0.00	0.00
20x20	0.00	0.00	0.00	0.00	0.00	0.00
50x5	1.42	1.41	1.92	1.91	0.50	0.51
50x10	2.41	2.39	2.81	2.81	0.39	0.39
50x20	2.53	2.55	3.08	3.07	0.54	0.53
100x5	2.40	2.35	5.28	5.23	2.81	2.80
100x10	3.95	3.93	6.44	6.39	2.40	2.36
100x20	4.09	4.10	6.13	6.15	1.96	2.00
200x10	5.22	5.24	8.83	8.94	3.43	3.47
200x20	5.37	5.44	8.78	8.84	3.24	3.21
500x20	7.11	7.09	10.74	10.72	3.39	3.46
Full set	2.87	2.60	4.60	4.60	1.59	1.37

Table 4.8: Results of BRKGA-Tuned and BRKGA-Random on the FSP.

Method	$ADev_B$	$MDev_B$
BRKGA-Tuned LS	2.87	2.60
BRKGA-Random NLS	3.28	3.45
BRKGA-Random LS	4.60	4.60
BRKGA-Tuned NLS	7.18	6.09

Table 4.9: Results of BRKGA-Random on TSP by instance group.

Problem	BRKGA TUNED LS		BRKGA RAND LS		RANDOM-TUNED RELATIVE	
	$ADev_B$	$MDev_B$	$ADev_B$	$MDev_B$	$ADev_T$	$MDev_T$
14 - 48	0.00	0.00	0.00	0.00	0.00	0.00
49 - 100	0.39	0.17	0.41	0.22	0.02	0.00
101 - 130	1.06	1.06	1.05	0.95	-0.02	0.00
131 - 198	1.78	1.59	1.82	1.61	0.04	0.02
199 - 318	3.86	3.57	3.40	3.50	-0.24	-0.11
319 - 575	19.18	7.49	6.39	6.63	-6.42	-0.82
576 - 1291	618.42	553.08	245.18	8.85	-56.57	-76.13
1292 - 2152	1863.99	1723.00	1673.31	1599.87	-5.54	-5.23
2153 - 5934	3805.61	2914.01	3686.85	2955.45	1.14	-2.09
	566.44	3.14	485.96	3.41	-10.02	-0.03

For the TSP with local search, the distribution of the deviations is less disperse than without local search. With BRKGA-Tuned LS having deviations very similar to BRKGA-Random LS, as can be observed in Figure 4.8.

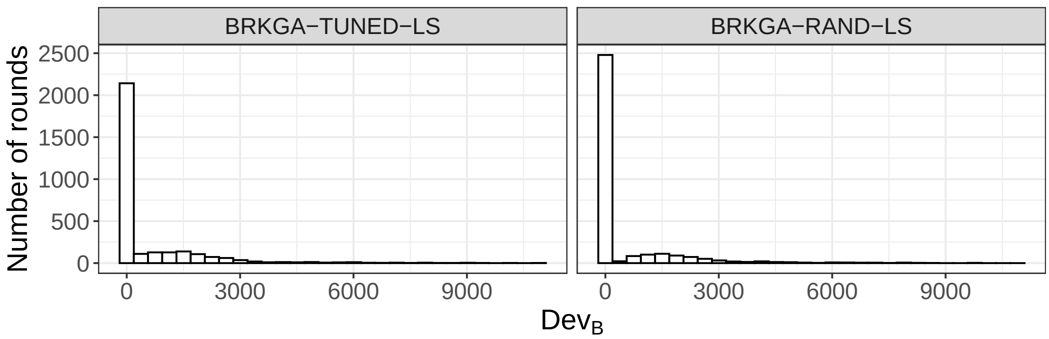

Figure 4.8: Distribution of relative percentage deviations from the best-known solution for the TSP instances (with Local Search).

Comparing the four methods for the TSP, regarding the average metric, BRKGA-Random LS led to the best results, followed by BRKGA-Tuned LS, BRKGA-Random NLS, and BRKGA-Tuned NLS as seen in Table 4.10.

Observing both TSP and FSP is possible to note the reduction in the performance of the random version compared to the tuned version (see $ADev_T$

Table 4.10: Results of BRKGA-Tuned and BRKGA-Random on the TSP.

Method	$ADev_B$	$MDev_B$
BRKGA-Random LS	485.96	3.41
BRKGA-Tuned LS	566.44	3.14
BRKGA-Random NLS	1607.68	1247.36
BRKGA-Tuned NLS	1934.32	908.77

metric on Tables 4.3 and 4.7 for the FSP, and Tables 4.4 and 4.9 for the TSP) when including the local searches to the decoders, which were once simple permutation algorithms. Hence, we suppose there is a relation between the decoder's complexity and the ability of the BRKGA with random parameter values to find good solutions.

In this case, we consider a decoder "more complex" if it presents variability within its execution. This means that for each execution of the decoder algorithm, we cannot know for sure how long it will take to decode a chromosome since it depends on the quality of the initial solution. The decoding process can be stuck on one solution for longer, and we can only estimate the best and worst-case scenarios. Since the random approach relies mostly on *exploration* we suppose that with simple decoders, such as the permutation ones, the method it is able to explore more of the solution space at a faster pace, allowing the evolution process of the BRKGA to find better solutions. Meanwhile, with Irace, the parameters are adjusted to the current decoder settings which may be why this effect is not perceived in the BRKGA-Tuned version.

For the SCP, however, the inclusion of the local search was beneficial to the overall solution quality and the relative performance of the BRKGA-Random, as can be observed in Table 4.11. The SCP already relied on a decoder with a greedy algorithm procedure to generate feasible solutions. By including an additional local search, we aimed to eliminate redundancies and produce better solutions in general. This addition does increase the decoder's complexity since it includes a step to the decoding procedure. But, in this case, the increase in solution quality had a higher impact than the increase in the decoding process complexity leading to better solutions obtained by the BRKGA-Random LS, by 0.15%.

Considering the SCP with the local search, the distribution of the deviations is much more dispersed than without the local search. This represents that with this method, more solutions with smaller deviations from the best-known solution were explored. BRKGA-Tuned LS and BRKGA-Random LS dispersion of deviations is very similar, as can be observed in Figure 4.9, with BRKGA-Random LS presenting slightly smaller values.

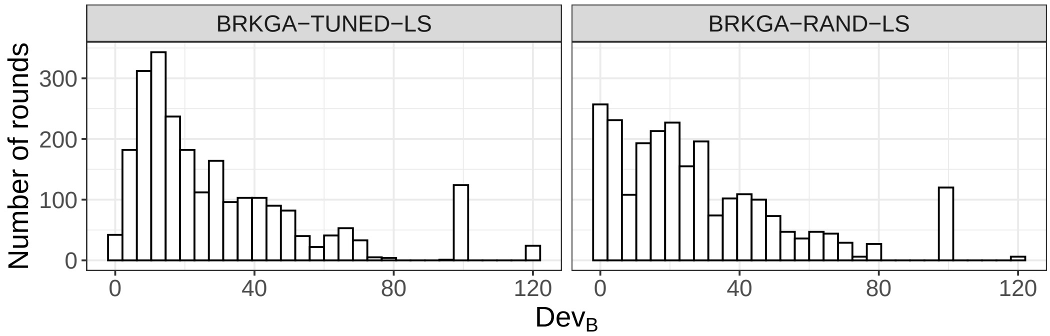

Figure 4.9: Distribution of relative percentage deviations from the best-known solution for the SCP instances (with Local Search).

Comparing the four methods for the SCP, BRKGA-Random LS led to the best results, followed by BRKGA-Tuned LS, BRKGA-Tuned NLS, and BRKGA-Random NLS as seen in Table 4.12.

Table 4.11: Results of BRKGA-Random with local search on SCP by instance group.

Problem	BRKGA TUNED LS		BRKGA RAND LS		RANDOM-TUNED RELATIVE	
	$ADev_B$	$MDev_B$	$ADev_B$	$MDev_B$	$ADev_T$	$MDev_T$
4	8.77	8.39	3.20	2.33	-5.01	-5.38
5	13.53	13.23	18.43	17.45	4.54	3.64
6	3.69	3.73	1.60	1.86	-2.00	-2.01
a	34.14	34.70	35.81	35.84	1.34	1.09
b	19.76	19.74	21.50	21.52	1.57	2.11
c	49.89	49.99	50.23	50.23	0.28	0.15
d	34.91	33.87	33.96	33.33	-0.59	0.00
e	102.68	100.00	97.60	100.00	-2.34	0.00
nre	20.98	21.43	21.10	21.43	0.18	0.00
nrf	12.79	14.29	13.06	14.29	0.23	0.00
nrg	64.77	65.41	64.22	64.25	-0.26	-0.35
nrh	42.09	42.86	42.51	42.62	0.36	0.00
Full Set	29.30	20.93	28.71	22.22	-0.15	0.00

Table 4.12: Results of BRKGA-Tuned and BRKGA-Random on the SCP.

Method	$ADev_B$	$MDev_B$
BRKGA-Random LS	28.70	22.20
BRKGA-Tuned LS	29.30	20.90
BRKGA-Tuned NLS	28430.00	1269.00
BRKGA-Random NLS	123503.00	17136.00

4.5 BRKGA-Race

After experimenting with random parameter values, we aimed to test whether incorporating a learning mechanism inspired in Irace [2] into BRKGA could tune parameters online and lead to similar results to tuning the algorithm beforehand. To investigate this hypothesis, we implemented the metaheuristic proposal described in Section 3.2 and conducted experiments similar to those described in the previous subsections.

We ran BRKGA-Race for the three problems, all problem instances, and 30 independent runs with a time limit of 3,600 seconds (1 hour), 7,200 seconds (2 hours), and 18,000 seconds (5 hours), considering the decoders with and without local search. The results of the race version are labeled as BRKGA-Race LS, for the decoder with local search, and BRKGA-Race NLS for the decoders without the local search. In this case, we extended the time limit to consider that the algorithm may need time to learn the best configurations and reach good solutions.

Before presenting all the experiments' results, we explore the evolution process of the BRKGA-Race in the following section, by examining an example case of the Flowshop Scheduling Problem.

4.6 BRKGA-Race: Example Case

In this section, we aim to bring a bit more clarity to the evolution process performed by BRKGA-Race. With that in mind, we selected a case to explore as an example. In Table 4.13, we can observe the output log of the execution of BRKGA-Race on the FSP's instance TA10, for one hour.

Each row of the table corresponds to one execution of BRKGA with a determined parameter setting (configuration), for the time budget stipulated for each evaluation on that race – which is called an "evaluation." The detail on what which column represents is detailed below.

- **Race**: The race id;
- **Conf**: The configuration id;
- p**,** p_e**,** p_m**, and** ρ_e: The assumed values the population size (p), the proportion of elite individuals (p_e), the proportion of mutant individuals (p_m), and the probability of inheriting a gene from parents from the elite set (ρ_e).;

Table 4.13: Output log of example case. BRKGA-Race execution on the FSP's instance TA10 for one hour.

race	config	p	p_e	p_m	ρ	initial cost	best cost	it	time budget	total elap. time	times eval.	pop. surv
1	1	4.08	0.18	0.49	0.53	13139	12968	1048	180	180	1	n
1	2	6.18	0.41	0.33	0.74	13183	12974	934	180	180	1	n
1	3	7.72	0.24	0.36	0.75	13267	12968	445	180	180	1	n
1	4	9.14	0.47	0.27	0.72	13126	12943	832	180	180	1	y
1	5	9.40	0.21	0.25	0.58	13094	12976	570	180	180	1	n
2	4	9.14	0.47	0.27	0.72	12943	12943	1000	225	219	2	y
2	6	4.22	0.16	0.34	0.54	13264	12968	1110	225	173	1	n
2	7	1.25	0.23	0.33	0.79	13272	12971	2446	225	107	1	n
2	8	5.76	0.35	0.31	0.69	13175	12976	1323	225	225	1	n
3	4	9.14	0.47	0.27	0.72	12943	12943	1000	300	213	3	y
3	9	3.36	0.21	0.27	0.73	13396	12943	1553	300	186	1	n
3	10	1.53	0.10	0.34	0.59	13077	12968	1845	300	117	1	n
4	4	9.14	0.47	0.27	0.72	12943	12943	1000	450	225	4	y
4	11	1.24	0.28	0.19	0.73	13202	13028	2024	450	163	1	n

- **Initial cost**: The cost of the initial solution produced by BRKGA (that is implemented without any warm start, so the initial solution recorded is the best solution obtained in the first generation of BRKGA);
- **Best cost**: The best solution cost obtained by the end of the evaluation;
- **Iterations (“it”)**: How many iterations of BRKGA were performed;
- **Time budget**: The given time budget for each evaluation;
- **Total elapsed time (“total elap. time”)**: How much time each evaluation actually lasted for;
- **Times evaluated (“times eval.”)**: How many times that configuration has been employed in a BRKGA run;
- **Pop Survives (“pop. surv”)**: If the population/configuration has survived in that race (y stands for “yes, it survived”, and n for “no”). This column is only populated at the end of the race.

In this execution of the BRKGA-Race, we were able to generate 11 new configurations and evaluate them while seeking a solution to the problem. One configuration survived along the way (Configuration 4) and was evaluated four times, seeking to improve the solution obtained by BRKGA configured with this particular setting. We performed 15 evaluations and four races with a total elapsed time of 2,545 seconds (of a total budget of 3,600 seconds, or 1 hour). The total time to perform the evaluations alone was of 2,528 seconds, and the other 17 seconds were employed in the parameter control procedures that were included in the BRKGA (generating new configurations, adapting configurations, selecting populations to survive, migrating individuals, etc.) –

which represents 0.06% of the total time. The minimum solution cost obtained in this run was 12,943 which is the optimal solution for this problem instance.

In the evaluations, BRKGA was able to perform 1,595 iterations on average. In these experiments, we included an additional stopping criteria to BRKGA to restrain it from running more than 1,000 iterations without improvement on the best solution. For that reason, it was possible to run the algorithm for less time than the allowed budget.

Observing the parameter values, we can see that they do not repeat themselves unless a population survives. See Configuration 4: after Race 1, this configuration survives. Then, it runs again on row 7. In this case, the values for all parameters remain the same. We run the same population, with the same settings, starting from where we left off. Note that the initial cost of row 7 is the best cost of row 4. And the column "Times Evaluated" indicates that this is the second time that configuration runs. By doing so, we aim to give a good configuration more time to improve its solution. This same configuration survives in the following two races, and even with the additional time given to it, it fails to improve the solution further, since it had already reached its optimal value.

The decision to survive is based upon the dominance criteria, explored in Section 3.2.3. At the end of a race, we evaluate each configuration against the other and establish the average dominance value for each configuration. Then we rank the configurations using the average dominance values and select a percentage p^s of the top configurations. In our experiments, we adopted a value of 20% for p^s, considering that a minimum of one population survives each race. This value was chosen after preliminary tests. As we can see in Table 4.13, we have five evaluations on Race 1, four evaluations on Race 2, three on Race 3, and two on Race 4. Since 20% of these numbers of evaluations never exceeds one, we end up with one surviving configuration in each race.

Returning to the parameter values, let's explore the adaptation process described in Section 3.2.4. In Race 1, parameter values are randomly generated within the intervals suggested in the literature, available in Table 2.2. From Race 2 forward, these values are adapted from the previously evaluated configurations and biased towards the ones with higher average dominance values. This bias is reinforced along with the races, as described in Equation (2-1). Table 4.14 illustrates the parent selection and the resulting adapted configuration from the studied example.

By calibrating the number of configurations being evaluated on each race, the time budget for each evaluation, the number of races, and the percentage of surviving populations we can influence the overall performance of

Table 4.14: Parent configuration and resulting adapted configurations.

(Race, Config)	Selected Parent (Race, Config)	Parent Configuration	Adapted Configuration
(2, 6)	(1, 1)	4.081, 0.180, 0.487, 0.530	4.223, 0.165, 0.340, 0.539
(3, 9)	(2, 7)	1.245, 0.233, 0.330, 0.792	3.356, 0.210, 0.275, 0.728
(4, 11)	(3, 9)	3.356, 0.210, 0.275, 0.728	1.241, 0.275, 0.193, 0.732

the algorithm. It is needed to balance the diversification of configurations, and the intensification of the best-found solutions. Due to time constraints, it was not possible to perform tuning of the algorithm itself and its hyperparameters in this work. For the tuning to be successful, it would need to incorporate multiple problems. Note that we do not aim to tailor this algorithm to one specific problem instance or class of problems, but to work well paired with the BRKGA framework while respecting (and even enhancing) BRKGA's characteristics.

4.7 BRKGA-Race: Results

The aggregated results for the three studied problems are displayed in Table 4.15. The table includes the average and median relative percentage deviations of both the Race version and the Random version, from the solution values obtained by BRKGA-Tuned (represented by the metrics $ADev_T$ and $MDev_T$, respectively). Full results comparing the solution values to the best-known solution for all methods implemented in this work are available in the Appendix B, C, and D.

It was possible to observe in the results that the increase in the time limit led to better solutions. In all three problems was possible to observe the reduction in the solutions' values when comparing the 1-hour run with the 5-hour run. Considering that this method eliminates the need for tuning,

Table 4.15: Results of BRKGA-Race and BRKGA-Random on FSP, TSP, and SCP.

Problem	Max time (hours)	BRKGA RACE LS		BRKGA RANDOM LS		BRKGA RACE NLS		BRKGA RANDOM NLS	
		$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$
FSP	1	-3.25	0.15	1.59	1.37	0.06	3.29	-3.57	-3.12
	2	-3.38	0.07	-	-	-1.24	1.88	-	-
	5	-3.53	0.01	-	-	-2.30	0.80	-	-
TSP	1	35.73	4.68	-15.40	-0.14	-0.96	-0.04	-46.54	-44.65
	2	33.34	4.09	-	-	-2.84	-0.49	-	-
	5	29.82	3.66	-	-	-5.98	-1.12	-	-
SCP	1	9.12	6.25	-0.15	0.00	1223.94	953.75	518.42	458.46
	2	7.11	4.26	-	-	985.89	741.73	-	-
	5	4.87	2.10	-	-	764.00	607.22	-	-

this increase in the execution time may be admissible. Especially in situations where there are no training instances available, or the overall time availability is shorter (considering that tuning might take several days) and fewer problem instances must be solved.

We did also observe a worse performance in the decoders with local search included, for the FSP and TSP. This might be because of the limited time available to evaluate each configuration. As detailed in Section 3.2, the time limit to evaluate/run one configuration starts at 180 seconds and increases by a fixed-rate percentage at each race. In our experiments, we set this rate at 25% due to the results observed in preliminary experiments. By adopting this setting, the final race configurations run for around seven minutes. When including the local search in the decoder, the decoding procedure's complexity increases and so does its execution time. For the SCP, as observed in the BRKGA-Random the inclusion of the local search is worth it.

It is important to recall that BRKGA, as a populational genetic algorithm, decodes multiple chromosomes at each generation. Associating a complex decoder with large problem instances may require more time to evolve a solution properly. Thus, the time available for each configuration might be insufficient. It is necessary, though, to balance the time spent on each configuration and the number of configurations being evaluated (diversification versus intensification) to obtain good results.

Table 4.16 displays results of the BRKGA-Race with and without local search, compared to the results of BRKGA-Random for the FSP. Both methods are evaluated against the solutions obtained by BRKGA-Tuned in the use case ($ADev_T$ and $MDev_T$). Looking at the results with local search (LS), and considering the execution time of 1 hour and the median metric ($MDev_T$), the Random version obtained better results in 5 instance groups (42%), equally good results in 4 instance groups (33%), and better results in 3 instance groups (25%). The cases where the Race version was better are the three groups with the largest instances. In these cases, BRKGA-Race was able to obtain solution values 16% lower than the solutions obtained with BRKGA-Tuned.

Considering the results without local search (NLS), the Race version did not present superior results for any instance group when compared to the Random version. When evaluating against the benchmark, the Tuned version, BRKGA-Race presented better solution values in the first three instance groups, when looking at the 5-hour runs. For the last three instance groups (the largest ones) it presented better results than the Tuned version for all execution times, providing solution values around 16% lower.

Figure 4.10 displays the behavior of the BRKGA-Race for FSP instance

Table 4.16: Results of BRKGA-Race on FSP.

Instance Group	Max Time (hours)	BRKGA RACE LS		BRKGA RANDOM LS		BRKGA RACE NLS		BRKGA RANDOM NLS	
		$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$
	1	0.02	0.00	0.00	0.00	1.49	1.56	-2.89	-2.64
20 x 5	2	0.01	0.00	-	-	0.27	0.33	-	-
	5	0.01	0.00	-	-	-0.70	-0.66	-	-
	1	0.00	0.00	0.00	0.00	1.55	1.74	-2.73	-2.69
20 x 10	2	0.00	0.00	-	-	0.34	0.20	-	-
	5	0.00	0.00	-	-	-0.56	-0.67	-	-
	1	0.01	0.00	0.00	0.00	0.99	1.01	-1.94	-1.93
20 x 20	2	0.00	0.00	-	-	0.15	0.21	-	-
	5	0.00	0.00	-	-	-0.38	-0.34	-	-
	1	1.30	1.33	0.50	0.51	7.21	7.40	-2.14	-2.10
50 x 5	2	1.16	1.21	-	-	5.30	5.46	-	-
	5	0.91	0.93	-	-	3.87	3.87	-	-
	1	1.00	1.02	0.39	0.39	6.52	6.65	-2.51	-2.46
50 x 10	2	0.82	0.82	-	-	4.99	5.11	-	-
	5	0.60	0.58	-	-	3.58	3.68	-	-
	1	1.00	1.00	0.54	0.53	5.55	5.60	-2.15	-2.15
50 x 20	2	0.76	0.78	-	-	4.11	4.32	-	-
	5	0.59	0.59	-	-	3.08	3.31	-	-
	1	3.15	3.10	2.81	2.80	7.97	8.15	-3.21	-3.44
100 x 5	2	2.96	2.90	-	-	6.36	6.22	-	-
	5	2.75	2.68	-	-	5.06	5.17	-	-
	1	2.60	2.54	2.40	2.36	7.92	8.09	-3.49	-3.46
100 x 10	2	2.40	2.36	-	-	6.60	6.87	-	-
	5	2.14	2.10	-	-	5.41	5.41	-	-
	1	2.03	2.00	1.96	2.00	6.94	6.97	-2.97	-2.74
100 x 20	2	1.82	1.84	-	-	5.80	5.86	-	-
	5	1.59	1.58	-	-	4.86	4.88	-	-
	1	-9.99	-9.97	3.43	3.47	-7.64	-7.62	-6.04	-6.07
200 x 10	2	-10.21	-10.21	-	-	-8.98	-8.86	-	-
	5	-10.35	-10.42	-	-	-9.96	-9.99	-	-
	1	-24.51	-24.41	3.24	3.21	-22.71	-22.55	-5.97	-6.04
200 x 20	2	-24.66	-24.55	-	-	-23.68	-23.62	-	-
	5	-24.63	-24.72	-	-	-24.76	-24.78	-	-
	1	-15.51	-15.64	3.39	3.46	-14.66	-14.61	-6.80	-6.84
500 x 20	2	-15.65	-15.79	-	-	-15.69	-15.49	-	-
	5	-15.87	-15.97	-	-	-16.48	-16.46	-	-

TA10. The same behavior was observed in the other instances, and this one was selected for illustration purposes. In the X-axis we observe the number of evaluations performed. One evaluation consists of one run of BRKGA with one configuration. When more time is given, more evaluations can be performed. We can see that in the first row of the chart: in a 1-hour run, the algorithm was able to evaluate BRKGA with 15 configurations, while in the 5-hour version it evaluated 73 configurations. In the Y-axis, we observe the progression of the current best solution cost.

The trend line on the 1-hour run is slightly turned upward, indicating that the solution cost is getting worse along with the evaluations. In this specific case, since we have a small number of configurations being evaluated on each race (5, 4, 3, and 2 configurations were evaluated on each one of the four races, respectively) only one configuration survived to the next race to be improved. Having only one surviving population indicates that the other ones are generated from new configurations, adapted from the previous ones.

We cannot guarantee that the generated configurations will lead to better solutions necessarily. The improvement in the solution values is observed when more time is given, allowing more configurations to be evaluated, survive, and incorporate learning from previous configurations into the next races. More details on BRKGA-Race internal procedures are available in Section 4.6.

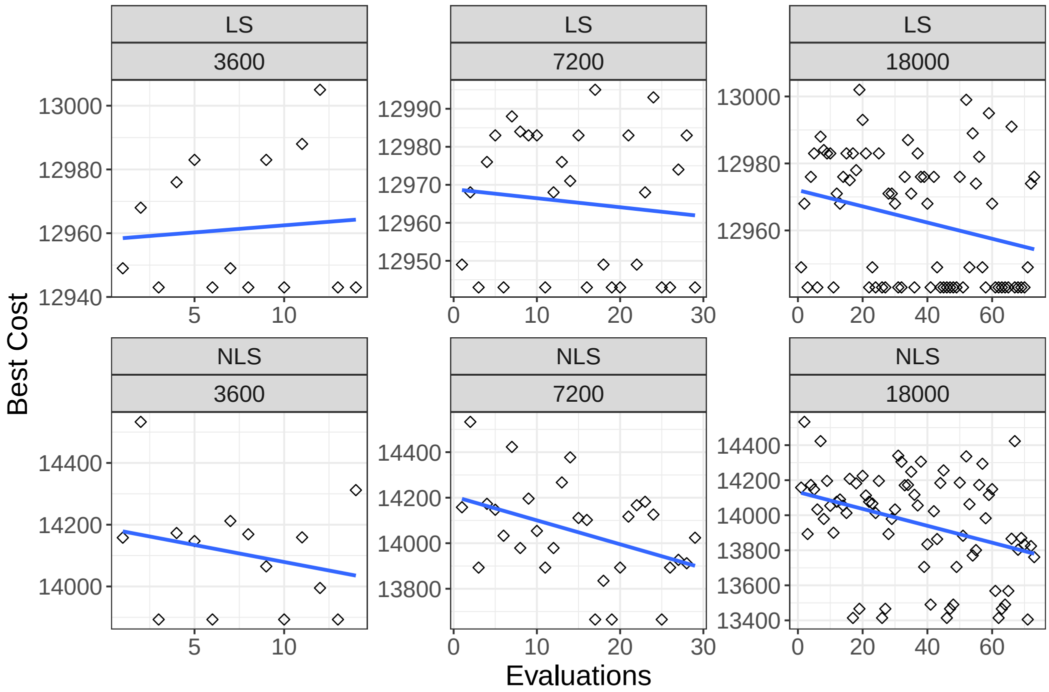

Figure 4.10: Evolution of the best cost throughout the evaluations of different configurations on the FSP (instance TA10).

We can observe that with more time, the trend line gets more accentuated towards lower solution values. This can point out the process of convergence of the algorithm. In the versions without the local search, this is subtly more noticeable. This might happen because the algorithm without local search generates worse solutions, but with more potential to improve faster without getting trapped in local optima.

The boxplots comparing the three evaluated methods on the FSP can be observed on Figures 4.11 and 4.12. Considering the decoder with local search, we can notice that the Race method presents similar results for 1-hour, 2-hour, and 5-hour execution times, with close median values and data distribution. The Random approach presents slightly worse results, and the Tuned version presents more normally distributed data around a smaller median value. Without local search, we can notice a higher increase in solution quality when adding more execution time to the Race approach. In this case, the Random distinguishes itself with better results.

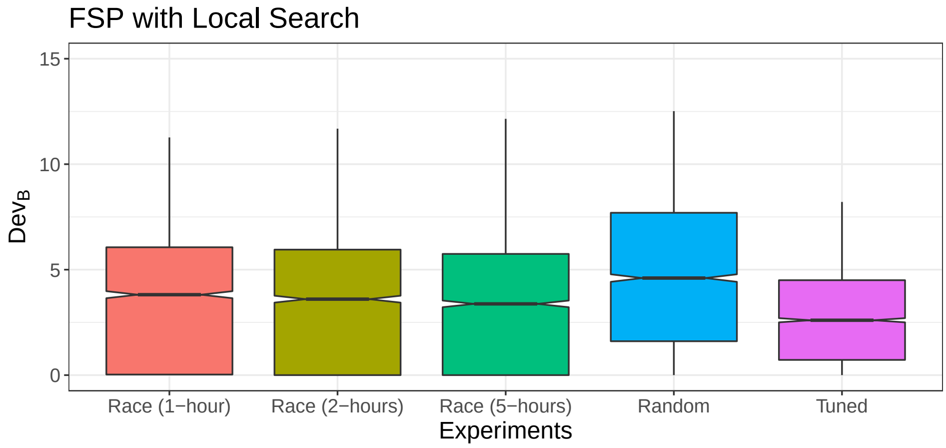

Figure 4.11: Boxplot comparing the deviations from the best known solutions of the three evaluated methods with local search on the FSP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race.

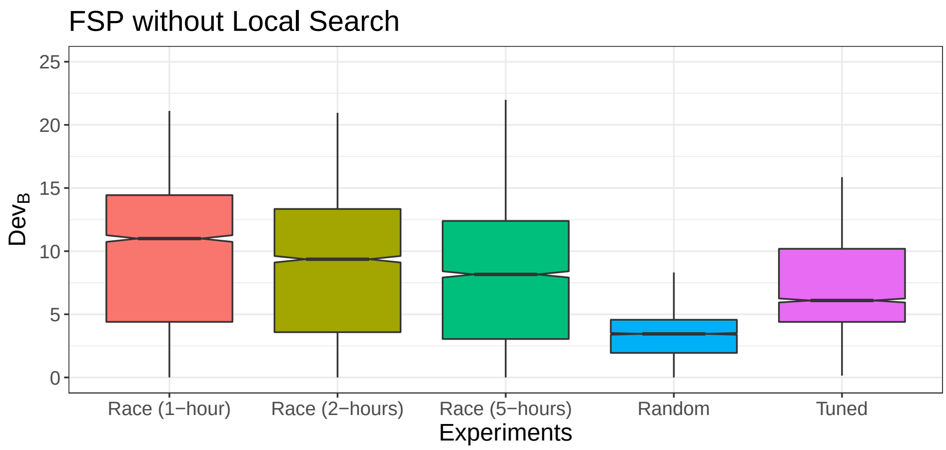

Figure 4.12: Boxplot comparing the deviations from the best known solutions of the three evaluated methods without local search on the FSP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race.

The results for the TSP can be observed in Table 4.17. In the table, we compare the results BRKGA-Race with and without local search, to the results of BRKGA-Random. Again, both methods are evaluated against the solutions obtained by BRKGA-Tuned in the use case ($ADev_T$ and $MDev_T$).

Observing the results for the 1-hour run without local search (NLS) we can see that BRKGA-Race was able to obtain better solutions than the tuned version in all instance groups when observing the average metric, and in 5 out of 9 instance groups when observing the median metric. When given more time, like in the 2-hour version, it was able to overcome the Tuned version in every instance group irregardless of the metric observed. When compared to the Random version though, it exhibited worse results (by around 58%) but was still superior to the benchmark.

The BRKGA-Race LS produced worse results when compared to the Tuned version. If we exclude instance groups 319-575 and 576-1291 and look at the median values, the results were around 0.07% worse than the benchmark solution costs. For the excluded instance groups, the results of BRKGA-Race LS were 117% worse. This case might happen due to the increased time in decoding when adding the local search. This group contains large instances, and decoding them with the 2-OPT local search might be taking too much time. Especially when considering the proposed method, which evaluates multiple configurations for shorter periods of time.

In the instance groups 319-575 and 576-1291 each evaluation ran for only one iteration in the Race version, and more iterations on the Tuned version. This means that in this group, the final solutions were actually the initial solutions that were generated randomly without going through the expected evolution in BRKGA. In the last instance group 2153-5934 both Race and Tuned versions only ran for one iteration. In other words, for these instance groups, the algorithm provided the best solution within a group of random solutions, which could indicate why the method presented these results. In the NLS version, this behavior is not observed – probably due to the lack of local search, allowing more iterations to be performed.

In Figure 4.13 we can see the evolution of the best solution cost alongside the configuration's evaluations for the TSP instance brazil58. As in the FSP, here we can also observe that the trend line becomes more accentuated with time. However, this effect is not as evident as it is in FSP.

The overall patterns of the three evaluated methods with and without local search on the TSP are displayed on Figures 4.14 and 4.15. Considering the version with local search, we can see that the Tuned version presents the best results, with more cohesive deviation values. Without local search, the Race version presents lower median values than the other methods, and the Tuned version presents more significant dispersion.

The results for the SCP can be observed in Table 4.18. Once again, we compare the results BRKGA-Race with and without local search, to the results of BRKGA-Random by observing the deviations from the solutions obtained by BRKGA-Tuned in the use case ($ADev_T$ and $MDev_T$).

The results of the BRKGA-Race LS were worse than those of the tuned version in the smaller instances sets (from "4" to "d"). However, in the more complex instance sets (from "c" to "nrh") the 5-hour runs outperformed both BRKGA-Random LS and BRKGA-Tuned LS. In the "nre", "nrf", "nrg", and "nrh" groups, BRKGA-Race LS outperformed BRKGA-Random LS in one hour, and BRKGA-Tuned LS in two hours.

Observing the results for the 1-hour run without local search (NLS) we can see that BRKGA-Race NLS's results outperformed BRKGA-Random LS's results in instance groups "c", "d", "nre", "nrf", "nrg", and "nrh" by a small percentage. Results from both approaches are not close to the tuned version ones. In the one-hour runs, results from BRKGA-Race NLS were 11.5x worse than results obtained by the algorithm tuned with irace. Considering 5-hour runs, the results are 7.3x of lower quality. But, without applying a local search to the SCP, even the results from the tuned version still fall far from the best-known solution.

Looking at Figure 4.16, we can observe the evolution of the best solution cost through the evaluations of the SCP instance scp55. A convergence trend can be perceived in all charts, as the solution cost decreases with the evaluations.

We can see the overview of all three evaluated methods with and without local search for the SCP on Figures 4.17 and 4.18. Considering the local search version, we can notice comparable results on all methods, with the Tuned version presenting lower median values. Without local search, the Race versions

Table 4.17: Results of BRKGA-Race on TSP.

Instance Group	Max Time (hours)	BRKGA RACE LS		BRKGA RANDOM LS		BRKGA RACE NLS		BRKGA RANDOM NLS	
		$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$
14 - 48	1	0.17	0.00	0.00	0.00	-3.81	-1.08	-37.81	-38.69
	2	0.12	0.00	-	-	-10.02	-5.46	-	-
	5	0.07	0.00	-	-	-15.93	-13.27	-	-
49 - 100	1	3.13	3.20	0.02	0.00	-1.46	0.59	-69.56	-71.64
	2	2.71	2.80	-	-	-5.87	-1.02	-	-
	5	2.43	2.45	-	-	-13.53	-3.32	-	-
101 - 130	1	4.16	4.15	-0.02	0.00	-0.84	0.31	-72.05	-71.76
	2	3.65	3.62	-	-	-3.03	-0.88	-	-
	5	3.23	3.30	-	-	-10.30	-2.52	-	-
131 - 198	1	4.44	4.69	0.04	0.02	-0.73	0.09	-73.35	-72.83
	2	3.86	4.11	-	-	-1.65	-0.56	-	-
	5	3.51	3.78	-	-	-6.66	-1.91	-	-
199 - 318	1	7.38	6.10	-0.24	-0.11	-0.23	-0.25	-67.86	-67.43
	2	7.23	5.65	-	-	-1.07	-0.78	-	-
	5	5.55	5.10	-	-	-3.65	-1.42	-	-
319 - 575	1	210.35	203.67	-6.42	-0.82	-0.21	-0.20	-56.37	-57.29
	2	204.65	194.14	-	-	-0.64	-0.54	-	-
	5	195.78	188.49	-	-	-1.25	-0.89	-	-
576 - 1291	1	64.52	46.04	-66.63	-84.23	-0.36	-0.08	-46.01	-46.64
	2	56.97	40.75	-	-	-0.83	-0.41	-	-
	5	47.92	32.07	-	-	-1.37	-0.69	-	-
1292 - 2152	1	2.35	0.19	-5.54	-5.23	-0.20	0.04	-37.78	-36.99
	2	0.50	-2.57	-	-	-0.48	-0.11	-	-
	5	-3.25	-6.02	-	-	-0.75	-0.35	-	-
2153 - 5934	1	-9.98	-10.24	1.14	-2.09	-0.42	-0.05	-21.12	-20.86
	2	-12.75	-13.74	-	-	-0.40	-0.11	-	-
	5	-15.04	-15.03	-	-	-0.59	-0.31	-	-

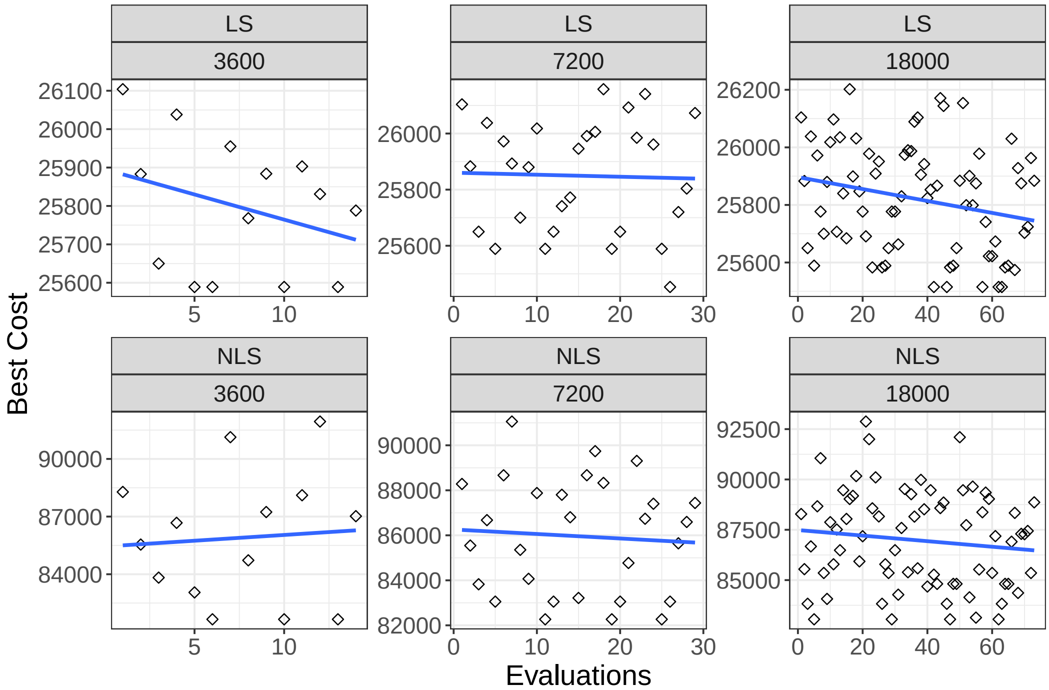

Figure 4.13: Evolution of the best cost throughout the evaluations of different configurations on the TSP (instance brazil58).

presents higher dispersion.

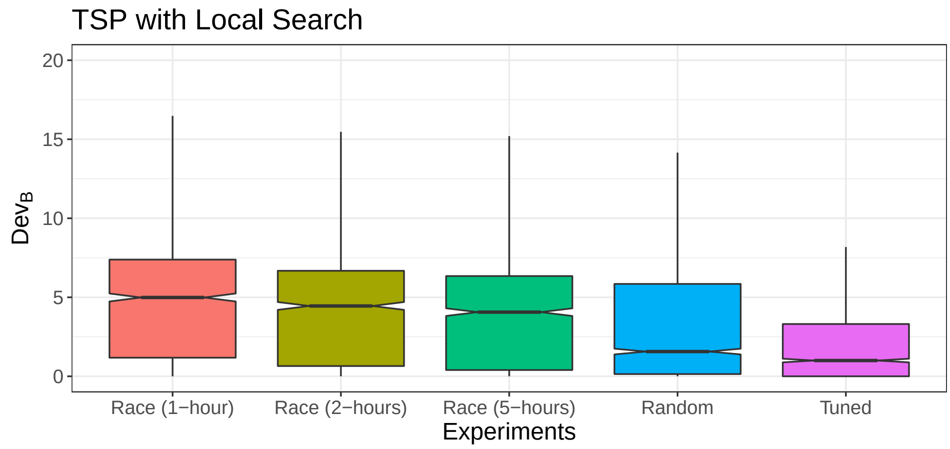

Figure 4.14: Boxplot comparing the deviations from the best known solutions of the three evaluated methods with local search on the TSP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race.

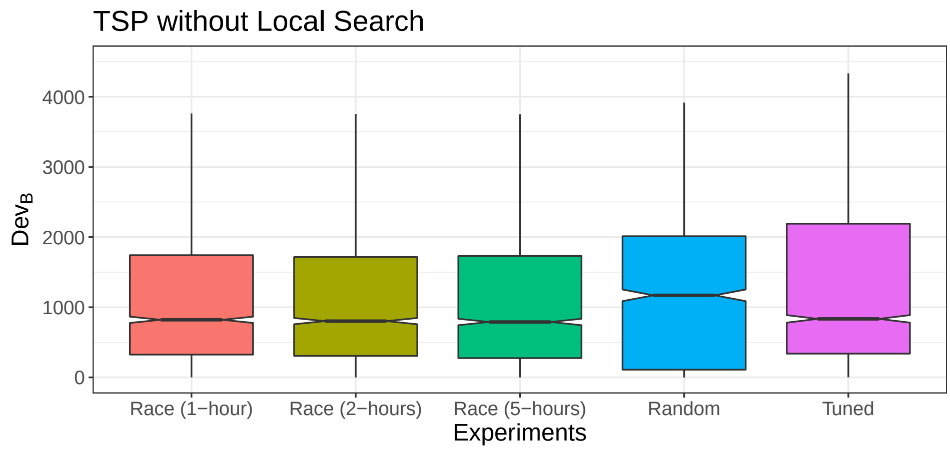

Figure 4.15: Boxplot comparing the deviations from the best known solutions of the three evaluated methods without local search on the TSP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race.

Table 4.18: Results of BRKGA-Race on SCP.

Instance Group	Max Time (hours)	BRKGA RACE LS		BRKGA RANDOM LS		BRKGA RACE NLS		BRKGA RANDOM NLS	
		$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$	$ADev_T$	$MDev_T$
4	1	20.36	20.23	-5.01	-5.38	1919.10	1921.78	170.63	146.38
	2	15.80	15.71	-	-	1373.10	1512.34	-	-
	5	11.62	11.85	-	-	999.33	927.23	-	-
5	1	24.80	24.18	4.54	3.64	1399.21	1403.22	461.12	435.52
	2	23.45	23.06	-	-	1131.49	1168.38	-	-
	5	20.78	20.57	-	-	836.32	860.29	-	-
6	1	9.93	9.02	-2.00	-2.01	4428.46	4426.96	387.84	254.92
	2	6.90	6.05	-	-	3435.55	3397.94	-	-
	5	4.73	4.39	-	-	2446.98	2058.79	-	-
a	1	9.59	9.45	1.34	1.09	1194.61	1210.40	1093.07	1068.15
	2	7.93	7.27	-	-	1012.40	1098.89	-	-
	5	6.54	6.37	-	-	821.02	817.70	-	-
b	1	9.22	8.57	1.57	2.11	1182.56	1185.77	1105.77	1102.90
	2	8.11	8.14	-	-	1048.15	1104.92	-	-
	5	5.11	5.32	-	-	870.10	884.70	-	-
c	1	2.94	3.13	0.28	0.15	850.15	873.20	909.47	894.43
	2	1.73	2.17	-	-	735.06	775.29	-	-
	5	-0.63	-0.56	-	-	664.69	688.24	-	-
d	1	3.37	3.39	-0.59	0.00	835.93	870.13	918.28	920.09
	2	1.48	1.20	-	-	766.57	795.20	-	-
	5	-1.01	0.00	-	-	670.13	697.91	-	-
e	1	-0.45	0.00	-2.34	0.00	559.75	550.00	-59.81	-62.50
	2	-1.28	0.00	-	-	378.30	346.15	-	-
	5	-2.28	0.00	-	-	195.42	205.88	-	-
nre	1	1.38	0.00	0.18	0.00	657.54	672.13	728.57	740.30
	2	-0.29	0.00	-	-	599.63	600.84	-	-
	5	-1.79	-2.78	-	-	558.84	585.85	-	-
nrf	1	0.60	0.00	0.23	0.00	635.84	633.39	710.79	709.01
	2	-0.40	0.00	-	-	585.27	590.15	-	-
	5	-1.88	0.00	-	-	535.58	541.64	-	-
nrg	1	-0.17	0.00	-0.26	-0.35	100.37	102.71	103.68	106.90
	2	-1.36	-1.44	-	-	101.22	99.80	-	-
	5	-2.95	-3.40	-	-	100.71	101.93	-	-
nrh	1	0.17	0.00	0.36	0.00	95.90	100.37	96.73	103.74
	2	-1.07	-1.17	-	-	95.06	103.93	-	-
	5	-2.28	-2.44	-	-	91.67	99.66	-	-

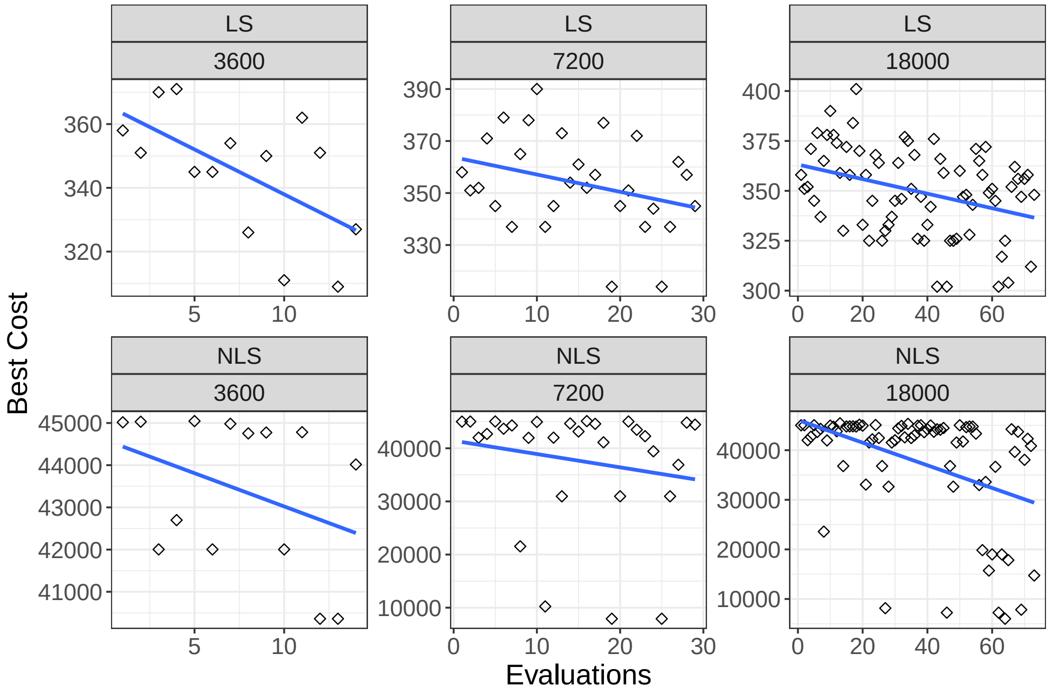

Figure 4.16: Evolution of the best cost throughout the evaluations of different configurations on the SCP (instance scp55).

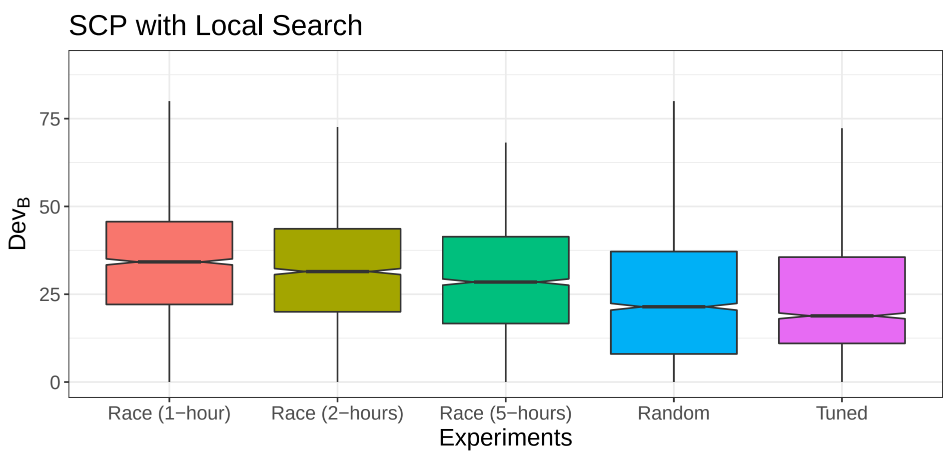

Figure 4.17: Boxplot comparing the deviations from the best known solutions of the three evaluated methods with local search on the SCP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race.

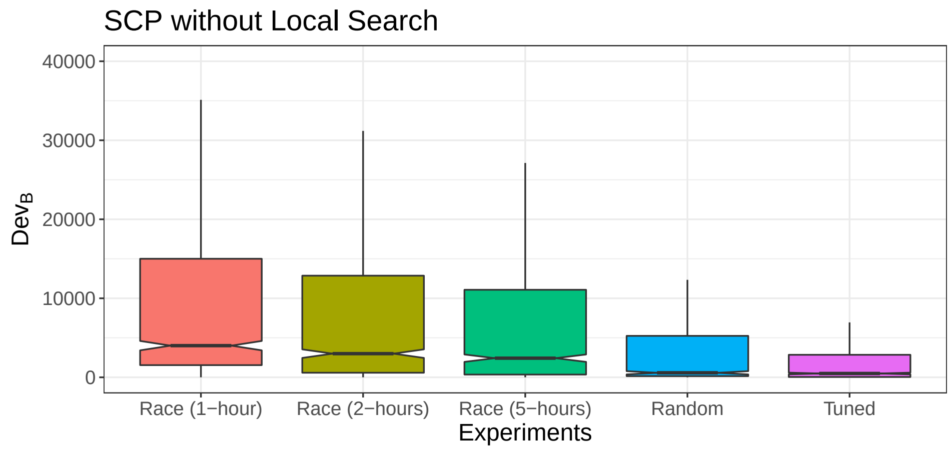

Figure 4.18: Boxplot comparing the deviations from the best known solutions of the three evaluated methods without local search on the SCP, considering 1-hour, 2-hours, and 5-hours of execution of BRKGA-Race.

5
Conclusions

In this work, we sought to propose and evaluate techniques to incorporate the concept of parameter control, or online tuning, into the Biased Random-key Genetic Algorithm (BRKGA) [1]. As described in the related literature accessed in this work, the BRKGA is a population-based metaheuristic that has multiple successful applications in complex optimization problems. Like GAs and other metaheuristics, BRKGA counts with several parameters that must be set before executing the algorithm, which turns the implementation more complex and time-consuming.

Online tuning is especially useful when dealing with problems without a proper training set, such as disaster relief scenarios, or when dealing with time constraints. Also, by simplifying and making the implementation of robust algorithms easier and faster we contribute to lowering the barriers to the scientific community to explore complex problems.

The Iterated Race (irace [2]) method is currently the state-of-the-art approach to parameter tuning, leading to good results when applied to BRKGA and others, as research showed. However, research on parameter control associated with BRKGA is still in its early days. In this case, we set the BRKGA tuned with irace as our benchmark for this work. Alongside, our goal with the proposed methods was to reach solution values close to those obtained by the algorithm tuned with Irace by employing less time and computational resources.

After assessing the literature and outlining the proposed methods, the first step of our experiments was setting our benchmark. The tuning process using irace proved itself to be very time-consuming and highly dependent on computational resources. In our experiments, the tuning step to generate our benchmark instances took from 3 days on the FSP without local search to 27 days on the FSP with local search to be completed. Since it is a long process, we had to deal with server crashes and memory outbreaks. We understand that the implementation of the algorithm being tuned impacts the irace performance, as well as the environment where it is being executed. But for this study, the experience reinforced the importance of researching different approaches and seeking to eliminate this step.

With the benchmark case in hand, we started the experiments with the two proposed approaches. First, we aimed to study the more straightforward method, in which we eliminated the need for tuning in BRKGA by adopting random parameter values. In this case, the user only needs to provide intervals for each parameter. We adopted values suggested in the literature for the upper and lower bounds of each parameter.

The results of the random approach demonstrated that this approach works very well with simple decoders, with faster execution time and no variability within executions. An example of this was shown when we included a local search to the decoder of the FSP, and it led to a 40% decrease in solution quality when compared to the decoder without local search. Similarly with the TSP, including a local search within the decoder led to solution values, on average, 10% better than the solutions obtained by BRKGA-Tuned LS. When comparing the random version without local search to the tuned version (also without local search) the solution values obtained by the first were 47% better. So, including a local search led to a decrease in the performance of the proposed method.

We suppose that with simple decoders, such as permutation ones – as is the case of the FSP and TSP decoders – the random approach allows BRKGA to explore more of the solution space at a faster pace. The random approach relies more on *exploration*, empowering the embedded mechanisms of BRKGA to test different approaches. Even though the complexity of decoders influenced the results of the random approach, the results of BRKGA-Random with Local Search were better than the results of the tuned version for the SCP (by 0.15%) and TSP (by 10.02%), and worse for the FSP (by 1.59%). Results are very competitive considering that no time was employed tuning the algorithm and that it ran for the same time (1 hour) as the tuned version.

After experimenting with random parameter values, we aimed to test whether incorporating a learning mechanism inspired by Irace could lead to better results. So we introduced the principles adopted by Irace in BRKGA, designing a metaheuristic (BRKGA-Race) that tunes itself while solving the optimization problem. Results show that BRKGA-Race presents competitive results when compared to the tuned algorithm. Considering the version with local search in the 1-hour execution and the median metric, the results for the FSP were 0.15% worse than the results obtained with the tuned version. For the TSP, they were 4.68% worse, and for SCP, 6.25%. In the BRKGA-Race approach, the values of the parameters were fixed during a race.

Giving more execution time to BRKGA-Race led to better solutions. In the three studied problems, it was possible to observe the reduction in the solutions' values when comparing the 1-hour run with the 5-hour run. In the 5-hour run, considering the version with local search, the results for the FSP were 0.01% worse than the results obtained with the tuned version. For the TSP, they were 3.66% worse, and for SCP, 2.10%. Considering that this method eliminates the need for tuning, this increase in the execution time may be admissible. Especially in situations where there are no training instances available, or the overall time availability is shorter (considering that tuning might take several days) and fewer problem instances must be solved.

BRKGA-Race also presented superior results for some of the larger instance groups. For the FSP, for example, in the last three instance groups (the largest ones) BRKGA-Race presented better results than the Tuned version for all execution times, providing solution values around 16% lower.

Due to time constraints when performing this work, we were not able to test multiple configurations and adjust the algorithm to different scenarios, by fine-tuning its hyperparameters. Note that we do not aim to tailor this algorithm to one specific problem instance or class of problems, but to work well paired with the BRKGA framework while respecting (and even enhancing) BRKGA's characteristics. So, in future works, we highlight the importance of adjusting the hyperparameters of the proposed method and testing it with other problems.

When comparing all methods adopted in this work, we rank the BRKGA-Random approach as the method with the overall best results, followed by BRKGA tuned offline with Irace, and lastly, the BRKGA-Race approach. We observed that changing parameter values - even if done randomly - throughout the BRKGA's generations can be highly beneficial to its performance. For future works, we suggest studying other approaches to adapt the parameter values during the algorithm's execution that incorporate knowledge of the evolution process, without leaving the values fixed along the way. In BRKGA-Race, even though the adaptation of parameter values was more sophisticated, we believe that leaving the values fixed through a full race had a negative impact. Also, we highlight to include in future works the explorations of different problems - especially those with more complex decoders.

6
References

[1] GONÇALVES, J. F.; RESENDE, M. G.. **Biased random-key genetic algorithms for combinatorial optimization**. Journal of Heuristics, 17(5):487–525, 2011. (document), 1, 2.1, 2.1, 2.1.1, 2.1.2, 2.2.1, 2.2, 3.1, 3.2.1, 4.3, 4.4, 5

[2] LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; CÁCERES, L. P.; BIRATTARI, M. ; STÜTZLE, T.. **The irace package: Iterated racing for automatic algorithm configuration**. Operations Research Perspectives, 3:43–58, 2016. (document), 1, 2.2.1, 2.2.1.1, 2.2, 2.2.1.1, 2.3, 2.2.1.1, 1, 9, 9, 3, 3.2, 3.2.1, 3.2.1, 3.2.2, 3.2.3, 3.2.4, 4, 4.3, 4.3, 4.5, 5

[3] ANDRADE, C.; SILVA, T. ; PESSOA, L.. **Minimizing flowtime in a flowshop scheduling problem with a biased random-key genetic algorithm**. Expert Systems with Applications, 128:67–80, 2019. (document), 1, 2.1.1, 2.1.2, 2.1, 2.2.1, 2.3, 4.1, 4.1.1, 4.1

[4] SÖRENSEN, K.; GLOVER, F.. **Metaheuristics**. Encyclopedia of Operations Research and Management Science, 2013. 1

[5] EIBEN, A.E., M. Z. S. M.; SMITH, J.. **Parameter control in evolutionary algorithms**. volumen 54 de **Studies in Computational Intelligence**, chapter 2, p. 19–46. Springer, 2007. 1, 2.2.1, 2.2.2

[6] HOOS, H. H.. **Automated algorithm configuration and parameter tuning**. p. 37–71. Springer, 2011. 1, 2.2

[7] KARAFOTIAS, G.; HOOGENDOORN, M. ; EIBEN, Á. E.. **Parameter control in evolutionary algorithms: Trends and challenges**. IEEE Transactions on Evolutionary Computation, 19(2):167–187, 2014. 1, 2.2.2, 3.2

[8] STEFANELLO, F.; AGGARWAL, V.; BURIOL, L. ; RESENDE, M.. **Hybrid algorithms for placement of virtual machines across geo-separated data centers**. Journal of Combinatorial Optimization, 38:748–793, 2019. 1, 2.1.2, 2.1, 2.2.1, 2.3, 3.2

[9] ABREU, L.; TAVARES-NETO, R. ; NAGANO, M.. **A new efficient biased random key genetic algorithm for open shop scheduling**

with routing by capacitated single vehicle and makespan minimization. Engineering Applications of Artificial Intelligence, 104, 2021. 1, 2.1.2, 2.1, 2.2.1, 2.3

[10] KUMMER, A.; BURIOL, L. ; ARAUJO, O. D.. **A Biased Random-key Genetic Algorithm applied to the VRPTW with skill requirements and synchronization constraints**. GECCO - 2020 - Proceedings of the 2020 Genetic and Evolutionary Computation Conference, p. 717–724, 2020. 1, 2.1.2, 2.1, 2.2.1, 2.3

[11] CUNHA, V.; PESSOA, L.; VELLASCO, M.; TANSCHEIT, R. ; PACHECO, M.. **A biased random-key genetic algorithm for the rescue unit allocation and scheduling problem**. 2018 IEEE Congress on Evolutionary Computation, CEC 2018 - Proceedings, 2018. 1, 2.1.2, 2.1, 2.2.1, 2.3

[12] MAURI, G.; BIAJOLI, F.; RABELLO, R.; CHAVES, A.; RIBEIRO, G. ; LORENA, L.. **Hybrid metaheuristics to solve a multiproduct two-stage capacitated facility location problem**. International Transactions in Operational Research, 28:3069–3093, 2021. 1, 2.1.2, 2.1, 2.2.1, 2.3, 4.1

[13] LONDE, M.; ANDRADE, C. ; PESSOA, L.. **An evolutionary approach for the p-next center problem**. Expert Systems with Applications, 175, 2021. 1, 2.1.2, 2.1, 2.2.1, 2.3, 4.1

[14] ANDRADE, C.; RESENDE, M.; ZHANG, W.; SINHA, R.; REICHMANN, K.; DOVERSPIKE, R. ; MIYAZAWA, F.. **A biased random-key genetic algorithm for wireless backhaul network design**. Applied Soft Computing Journal, 33:150–169, 2015. 1, 2.1.2, 2.1, 2.2.1, 2.3

[15] PINTO, B.; RIBEIRO, C.; RIVEAUX, J. ; ROSSETI, I.. **A BRKGA-based matheuristic for the maximum quasi-clique problem with an exact local search strategy**. RAIRO - Operations Research, 55:S741–S763, 2021. 1, 2.1.2, 2.1, 2.3, 4.1

[16] PESSOA, L. S.; ANDRADE, C. E.. **Heuristics for a flowshop scheduling problem with stepwise job objective function**. European Journal of Operational Research, 266(3):950–962, 2018. 1, 2.1.2, 2.1, 2.3

[17] ANDRADE, C.; TOSO, R.; GONÇALVES, J. ; RESENDE, M.. **The multi-parent biased random-key genetic algorithm with implicit path-relinking and its real-world applications**. European Journal of

Operational Research, 289:17–30, 2021. 1, 2.1, 2.1.1, 2.1.2, 2.1, 2.2.1, 2.3, 4.1.2, 4.2

[18] GOLBERG, D. E.. **Genetic algorithms in search, optimization, and machine learning**. Addion-Wesley, 1989(102):36, 1989. 2.1

[19] WHITLEY, D.. **A genetic algorithm tutorial**. Statistics and Computing, 4(2):65–85, 1994. 2.1

[20] BEAN, J. C.. **Genetic algorithms and random keys for sequencing and optimization**. ORSA journal on computing, 6(2):154–160, 1994. 2.1

[21] FOGEL, D. B.. **An introduction to simulated evolutionary optimization**. IEEE transactions on neural networks, 5(1):3–14, 1994. 2.1.1

[22] RIBEIRO, C.; RIVEAUX, J. ; BRANDAO, J.. **Biased random-key genetic algorithms using path-relinking as a progressive crossover strategy**. ACM International Conference Proceeding Series, p. 28–36, 2021. 2.1.1, 2.1.2, 2.1, 2.2.1, 2.3

[23] ALBA, E.; TROYA, J. M.. **A survey of parallel distributed genetic algorithms**. Complexity, 1999. 2.1.1

[24] GONÇALVES, J. F.; RESENDE, M. G.. **A parallel multi-population biased random-key genetic algorithm for a container loading problem**. Computers Operations Research, 39(2):179–190, 2012. 2.1.1, 2.1.2, 2.1, 2.2.1, 2.3

[25] FARIA, H. D.; LUNARDI, W. ; VOOS, H.. **A parallel multi-population biased random-key genetic algorithm for electric distribution network reconfiguration**. GECCO 2019 Companion - Proceedings of the 2019 Genetic and Evolutionary Computation Conference Companion, p. 281–282, 2019. 2.1.1, 2.1.2, 2.1, 2.2.1, 2.3

[26] JÚNIOR, B. A.; PINHEIRO, P. ; COELHO, P.. **A parallel biased random-key genetic algorithm with multiple populations applied to irregular strip packing problems**. Mathematical Problems in Engineering, 2017, 2017. 2.1.1, 2.1.2, 2.1, 2.3

[27] ALIXANDRE, B.; DORN, M.. **D-BRKGA: A distributed biased random-key genetic algorithm**. 2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings, p. 1398–1405, 2017. 2.1.1, 2.1, 2.3

[28] OLIVEIRA, B.; CARRAVILLA, M.; OLIVEIRA, J. ; RESENDE, M.. **A C++ application programming interface for co-evolutionary biased random-key genetic algorithms for solution and scenario generation**. Optimization Methods and Software, 2021. 2.1.1, 2.1, 2.3

[29] BIAJOLI, F.; CHAVES, A. ; LORENA, L.. **A biased random-key genetic algorithm for the two-stage capacitated facility location problem**. Expert Systems with Applications, 115:418–426, 2019. 2.1.2, 2.2.1, 2.1, 2.2.1, 2.3, 4.1

[30] GONÇALVES, J.; RESENDE, M.. **A biased random-key genetic algorithm for the unequal area facility layout problem**. European Journal of Operational Research, 246:86–107, 2015. 2.1.2, 2.1, 2.2.1, 2.3

[31] LALLA-RUIZ, E.; EXPÓSITO-IZQUIERDO, C.; MELIÁN-BATISTA, B. ; MORENO-VEGA, J.. **A hybrid biased random key genetic algorithm for the quadratic assignment problem**. Information Processing Letters, 116:513–520, 2016. 2.1.2, 2.1, 2.3

[32] PESSOA, L.; SANTOS, A. ; RESENDE, M.. **A biased random-key genetic algorithm for the tree of hubs location problem**. Optimization Letters, 11:1371–1384, 2017. 2.1.2, 2.1, 2.2.1, 2.3

[33] CICEK, Z. I. E.; OZTURK, Z. K.. **Optimizing the artificial neural network parameters using a biased random key genetic algorithm for time series forecasting**. Applied Soft Computing, 102:107091, 2021. 2.1.2, 2.1, 2.2.1, 2.3

[34] CARRABS, F.. **A biased random-key genetic algorithm for the set orienteering problem**. European Journal of Operational Research, 292:830–854, 2021. 2.1.2, 2.1, 2.2.1, 2.3

[35] ZUDIO, A.; DA SILVA COSTA, D.; MASQUIO, B.; COELHO, I. ; PINTO, P.. **BRKGA/VND hybrid algorithm for the classic three-dimensional bin packing problem**. Electronic Notes in Discrete Mathematics, 66:175–182, 2018. 2.1.2, 2.1, 2.2.1, 2.3

[36] ROCHMAN, A.; PRASETYO, H. ; NUGROHO, M.. **Biased random key genetic algorithm with insertion and gender selection for capacitated vehicle routing problem with time windows**. AIP Conference Proceedings, 1855, 2017. 2.1.2, 2.1, 2.2.1, 2.3

[37] DAMM, R.; RESENDE, M. ; RONCONI, D.. **A biased random key genetic algorithm for the field technician scheduling problem**. Computers and Operations Research, 75:49–63, 2016. 2.1.2, 2.1, 2.3

[38] CHAVES, A.; LORENA, L.; SENNE, E. ; RESENDE, M.. **Hybrid method with CS and BRKGA applied to the minimization of tool switches problem**. Computers and Operations Research, 67:174–183, 2016. 2.1.2, 2.1, 2.2.1, 2.3

[39] AMARO, B.; PINHEIRO, P.. **μ-brkga: A parallel biased random-key genetic algorithm applied to nesting problems**. Proceedings of International Conference on Computers and Industrial Engineering, CIE, 2017. 2.1.2, 2.1, 2.2.1, 2.3

[40] GONÇALVES, J.. **A hybrid biased random key genetic algorithm for a production and cutting problem**. IFAC-PapersOnLine, 28:496–500, 2015. 2.1.2, 2.1, 2.2.1, 2.3

[41] HUANG, C., L. Y.; YAO, X.. **A survey of automatic parameter tuning methods for metaheuristics**. IEEE Transactions on Evolutionary Computation, 24(2):201–214, April 2020. 2.2, 2.2.1

[42] EIBEN, A.; HINTERDING, R. ; MICHALEWICZ, Z.. **Parameter control in evolutionary algorithms**. IEEE Transactions on Evolutionary Computation, 3(2):124–141, 1999. 2.2, 2.2.1, 2.2.2

[43] FEURER, M.; HUTTER, F.. **Hyperparameter optimization**. Automated Machine Learning, p. 3–33, 2019. 2.2.1

[44] KARIMI-MAMAGHAN, M.; MOHAMMADI, M.; MEYER, P.; KARIMI-MAMAGHAN, A. M. ; TALBI, E.-G.. **Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art**. European Journal of Operational Research, 2021. 2.2.1

[45] WOLPERT, D.; MACREADY, W.. **No free lunch theorems for optimization**. IEEE Transactions on Evolutionary Computation, 1(1):67–82, 1997. 2.2.1

[46] BIRATTARI, M.. **F-Race for Tuning Metaheuristics**, p. 85–115. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. 2.2.1

[47] ADENSO-DÍAZ, B.; LAGUNA, M.. **Fine-tuning of algorithms using fractional experimental designs and local search**. Operations Research, 54:99–114, 02 2006. 2.2.1

[48] BIRATTARI, M.; YUAN, Z.; BALAPRAKASH, P. ; STÜTZLE, T.. **F-Race and Iterated F-Race: An Overview**, p. 311–336. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. 2.2.1, 2.2.1.1

[49] MERCER, R.; SAMPSON, J.. **Adaptive search using a reproductive metaplan**. Kybernetes, 7:215–228, 1978. 2.2.1

[50] HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. ; STÜTZLE, T.. **ParamILS: an automatic algorithm configuration framework**. Journal of Artificial Intelligence Research, 36:267–306, 2009. 2.2.1

[51] LÓPEZ-IBÁÑEZ, M.; CÁCERES, L. P.; DUBOIS-LACOSTE, J.; STÜTZLE, T. G. ; BIRATTARI, M.. **The irace package: User guide**. IRIDIA, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle, 2016. 2.2.1.1

[52] CONOVER, W. J.. **Practical nonparametric statistics**, volumen 350. John Wiley & Sons, 1999. 9, 3.2.3

[53] SEVAUX, M.; SÖRENSEN, K. ; PILLAY, N.. **Adaptive and Multilevel Metaheuristics**, p. 1–19. Springer International Publishing, Cham, 2018. 2.2.2

[54] ARABAS, J.; MICHALEWICZ, Z. ; MULAWKA, J.. **GAVaPS-a genetic algorithm with varying population size**. Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence, p. 73–78 vol.1, 1994. 2.2.2

[55] HINTERDING, R.; MICHALEWICZ, Z. ; PEACHEY, T. C.. **Self-adaptive genetic algorithms for numeric functions**. 1996. 2.2.2

[56] BÄCK, T.; EIBEN, A. E. ; VAN DER VAART, N. A.. **An empirical study on GAs "without parameters"**. International Conference on Parallel Problem Solving from Nature, p. 315–324, 2000. 2.2.2

[57] EIBEN, A. E.; MARCHIORI, E. ; VALKO, V.. **Evolutionary algorithms with on-the-fly population size adjustment**. International Conference on Parallel Problem Solving from Nature, p. 41–50, 2004. 2.2.2

[58] ALETI, A.; MOSER, I.; MEEDENIYA, I. ; GRUNSKE, L.. **Choosing the appropriate forecasting model for predictive parameter control**. Evolutionary Computation, 22:319–349, 2014. 2.2.2

[59] CHAVES, A.; GONÇALVES, J. ; LORENA, L.. **Adaptive biased random-key genetic algorithm with local search for the capacitated centered clustering problem**. Computers and Industrial Engineering, 124:331–346, 2018. 2.2.2, 2.3

[60] CHAVES, A. A.; LORENA, L. H. N.. **An adaptive and near parameter-free BRKGA using Q-learning method**. 2021 IEEE Congress on Evolutionary Computation (CEC), p. 2331–2338, 2021. 2.2.2, 2.3

[61] SCHUETZ, M. J.; BRUBAKER, J. K.; MONTAGU, H.; VAN DIJK, Y.; KLEPSCH, J.; ROSS, P.; LUCKOW, A.; RESENDE, M. G. ; KATZGRABER, H. G.. **Optimization of robot trajectory planning with nature-inspired and hybrid quantum algorithms**. arXiv preprint arXiv:2206.03651, 2022. 2.2.2, 2.3

[62] BERGSTRA, J.; YAMINS, D. ; COX, D.. **Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 115–123. PMLR, 2013. 2.2.2

[63] GAUCH JR, H. G.. **Scientific method in practice**. Cambridge University Press, 2003. 3.1

[64] QUIRK, J. P.; SAPOSNIK, R.. **Admissibility and measurable utility functions**. The Review of Economic Studies, 29(2):140–146, 1962. 3.2.3

[65] MOSCATO, P.. **On evolution, search, optimization, gas and martial arts: toward memetic algorithms. california inst. technol., pasadena**. Technical report, CA, Tech. Rep. Caltech Concurrent Comput. Prog. Rep. 826, 1989. 4.1

[66] TAILLARD, E.. **Some efficient heuristic methods for the flow-shop sequencing problem**. European Journal of Operational Research, 47(1):65–74, 1990. 4.1.1

[67] DEN BESTEN, M.; STÜTZLE, T.. **Neighborhoods revisited: An experimental investigation into the effectiveness of variable neighborhood descent for scheduling**. Proceedings of the 4th Metaheuristics International Conference (MIC-01), p. 545–550, 2001. 4.1.1

[68] TAILLARD, E.. **Benchmarks for basic scheduling problems**. European Journal of Operational Research, 64(2):278–285, 1993. 4.1.1

[69] GRECO, F.. **Traveling Salesman Problem**. IntechOpen, 2008. 4.1.2

[70] CROES, G. A.. **A method for solving traveling-salesman problems**. Operations Research, 6(6):791–812, 1958. 4.1.2

[71] REINELT, G.. **TSPLIB — a Traveling Salesman Problem library**. ORSA Journal on Computing, 3(4):376–384, 1991. 4.1.2

[72] BEASLEY, J. E.. **An algorithm for Set Covering Problem**. European Journal of Operational Research, 31(1):85–93, 1987. 4.1.3, 4.1.3

[73] BEASLEY, J. E.. **A lagrangian heuristic for Set-Covering Problems**. Naval Research Logistics (NRL), 37(1):151–164, 1990. 4.1.3, 4.1.3

[74] JOHNSON, D. S.. **Approximation algorithms for combinatorial problems**. Journal of Computer and System Sciences, 9(3):256–278, 1974. 4.1.3

[75] FEO, T. A.; RESENDE, M. G.. **A probabilistic heuristic for a computationally difficult Set Covering Problem**. Operations Research Letters, 8(2):67–71, 1989. 4.1.3

[76] BEASLEY, J. E.. **OR-Library: distributing test problems by electronic mail**. Journal of the Operational Research Society, 41(11):1069–1072, 1990. 4.1.3

[77] BEASLEY, J. E.. **OR-Library**. `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html`. 4.1.3, 4.4

[78] BEZANSON, J.; EDELMAN, A.; KARPINSKI, S. ; SHAH, V. B.. **Julia: A fresh approach to numerical computing**. SIAM review, 59(1):65–98, 2017. 4.2

A
Instance's Dimensions

A.1
Flowshop Scheduling Problem

Table A.1: The table presents the dimensions for the FSP instances used in this work.

Instance	Jobs (n)	Machines (m)
TA1	20	5
TA2	20	5
TA3	20	5
TA4	20	5
TA5	20	5
TA6	20	5
TA7	20	5
TA8	20	5
TA9	20	5
TA10	20	5
TA11	20	10
TA12	20	10
TA13	20	10
TA14	20	10
TA15	20	10
TA16	20	10
TA17	20	10
TA18	20	10
TA19	20	10
TA20	20	10
TA21	20	20
TA22	20	20
TA23	20	20
TA24	20	20
TA25	20	20
TA26	20	20
TA27	20	20
TA28	20	20
TA29	20	20
TA30	20	20
TA31	50	5
TA32	50	5
TA33	50	5
TA34	50	5
TA35	50	5
TA36	50	5
TA37	50	5
TA38	50	5
TA39	50	5
TA40	50	5
TA41	50	10
TA42	50	10
TA43	50	10
TA44	50	10
TA45	50	10
TA46	50	10

Instance	Jobs (n)	Machines (m)
TA47	50	10
TA48	50	10
TA49	50	10
TA50	50	10
TA51	50	20
TA52	50	20
TA53	50	20
TA54	50	20
TA55	50	20
TA56	50	20
TA57	50	20
TA58	50	20
TA59	50	20
TA60	50	20
TA61	100	5
TA62	100	5
TA63	100	5
TA64	100	5
TA65	100	5
TA66	100	5
TA67	100	5
TA68	100	5
TA69	100	5
TA70	100	5
TA71	100	10
TA72	100	10
TA73	100	10
TA74	100	10
TA75	100	10
TA76	100	10
TA77	100	10
TA78	100	10
TA79	100	10
TA80	100	10
TA81	100	20
TA82	100	20
TA83	100	20
TA84	100	20
TA85	100	20
TA86	100	20
TA87	100	20
TA88	100	20
TA89	100	20
TA90	100	20
TA91	200	10
TA92	200	10
TA93	200	10
TA94	200	10
TA95	200	10
TA96	200	10
TA97	200	10
TA98	200	10
TA99	200	10
TA100	200	10
TA101	200	20
TA102	200	20
TA103	200	20
TA104	200	20
TA105	200	20
TA106	200	20

Instance	Jobs (n)	Machines (m)
TA107	200	20
TA108	200	20
TA109	200	20
TA110	200	20
TA111	500	20
TA112	500	20
TA113	500	20
TA114	500	20
TA115	500	20
TA116	500	20
TA117	500	20
TA118	500	20
TA119	500	20
TA120	500	20

A.2 Set Covering Problem

Table A.2: The table presents the dimensions for the SCP instances used in this work.

Instance	Objects (m)	Subsets (n)
scp41	200	1000
scp42	200	1000
scp43	200	1000
scp44	200	1000
scp45	200	1000
scp46	200	1000
scp47	200	1000
scp48	200	1000
scp49	200	1000
scp51	200	2000
scp52	200	2000
scp53	200	2000
scp54	200	2000
scp55	200	2000
scp56	200	2000
scp57	200	2000
scp58	200	2000
scp59	200	2000
scp61	200	1000
scp62	200	1000
scp63	200	1000
scp64	200	1000
scp65	200	1000
scp410	200	1000
scp510	200	2000
scpa1	300	3000
scpa2	300	3000
scpa3	300	3000
scpa4	300	3000
scpa5	300	3000
scpb1	300	3000
scpb2	300	3000
scpb3	300	3000
scpb4	300	3000

Instance	Objects (m)	Subsets (n)
scpb5	300	3000
scpc1	400	4000
scpc2	400	4000
scpc3	400	4000
scpc4	400	4000
scpc5	400	4000
scpd1	400	4000
scpd2	400	4000
scpd3	400	4000
scpd4	400	4000
scpd5	400	4000
scpe1	50	500
scpe2	50	500
scpe3	50	500
scpe4	50	500
scpe5	50	500
scpnre1	500	5000
scpnre2	500	5000
scpnre3	500	5000
scpnre4	500	5000
scpnre5	500	5000
scpnrf1	500	5000
scpnrf2	500	5000
scpnrf3	500	5000
scpnrf4	500	5000
scpnrf5	500	5000
scpnrg1	1000	10000
scpnrg2	1000	10000
scpnrg3	1000	10000
scpnrg4	1000	10000
scpnrg5	1000	10000
scpnrh1	1000	10000
scpnrh2	1000	10000
scpnrh3	1000	10000
scpnrh4	1000	10000
scpnrh5	1000	10000

B
Complete Results for the FSP

Table B.1: The table presents the complete results for the FSP without Local Search. Each row represents a problem instance, and "BKS" stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature.

Instance	BKS	Tuned-NLS (1-hour)			Random-NLS (1-hour)			Race-NLS (1-hour)			Race-NLS (2-hours)			Race-NLS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
TA1	14,033	14,421	2.76	2.74	14,083	0.36	0.33	14,641	4.33	4.05	14,496	3.30	3.44	14,360	2.33	2.28
TA2	15,151	15,624	3.12	3.04	15,230	0.52	0.42	15,811	4.36	4.28	15,707	3.67	3.92	15,598	2.95	2.75
TA3	13,301	13,914	4.61	4.70	13,420	0.89	0.86	14,138	6.29	6.46	13,943	4.83	4.97	13,812	3.84	3.98
TA4	15,447	15,935	3.16	3.05	15,581	0.86	0.92	16,144	4.51	4.59	15,978	3.44	3.41	15,813	2.37	2.20
TA5	13,529	13,878	2.58	2.34	13,571	0.31	0.35	14,089	4.14	3.85	13,915	2.86	2.47	13,790	1.93	1.78
TA6	13,123	13,713	4.50	4.22	13,191	0.52	0.44	13,974	6.49	6.52	13,725	4.59	3.69	13,549	3.25	3.23
TA7	13,548	14,131	4.31	4.27	13,714	1.22	1.00	14,314	5.65	5.81	14,140	4.37	4.11	13,990	3.27	3.18
TA8	13,948	14,480	3.82	3.53	14,005	0.41	0.35	14,751	5.76	5.85	14,498	3.94	3.53	14,328	2.72	2.11
TA9	14,295	14,947	4.56	4.54	14,426	0.92	0.70	15,108	5.69	6.04	14,954	4.61	4.58	14,830	3.74	3.85
TA10	12,943	13,437	3.82	3.65	13,071	0.99	1.05	13,642	5.40	5.05	13,526	4.50	5.01	13,368	3.29	3.14
TA11	20,911	21,710	3.82	3.73	21,063	0.73	0.54	22,085	5.62	5.82	21,830	4.40	4.71	21,589	3.24	2.98
TA12	22,440	23,298	3.82	3.80	22,712	1.21	1.58	23,610	5.22	5.26	23,435	4.44	4.02	23,241	3.57	3.54
TA13	19,833	20,542	3.57	3.32	19,951	0.59	0.63	20,795	4.85	5.60	20,481	3.27	2.77	20,345	2.58	2.36
TA14	18,710	19,417	3.78	3.76	18,896	1.00	0.89	19,815	5.90	6.27	19,500	4.22	4.32	19,385	3.61	3.62
TA15	18,641	19,272	3.39	3.12	18,761	0.64	0.57	19,614	5.22	5.64	19,389	4.01	3.64	19,164	2.80	2.81
TA16	19,245	19,883	3.32	3.23	19,421	0.91	0.79	20,150	4.70	4.83	19,970	3.77	3.89	19,822	3.00	2.94
TA17	18,363	19,070	3.85	3.84	18,468	0.57	0.46	19,348	5.36	5.55	19,059	3.79	3.76	18,887	2.86	2.52
TA18	20,241	21,018	3.84	3.77	20,394	0.76	0.73	21,433	5.89	6.38	21,171	4.59	4.75	20,975	3.62	3.13
TA19	20,330	21,092	3.75	3.82	20,474	0.71	0.57	21,317	4.85	4.85	21,056	3.57	3.47	20,871	2.66	2.55
TA20	21,320	21,958	2.99	3.03	21,444	0.58	0.59	22,295	4.57	4.87	22,045	3.40	3.68	21,885	2.65	2.38
TA21	33,623	34,582	2.85	2.93	33,901	0.83	0.76	34,927	3.88	3.86	34,706	3.22	3.32	34,517	2.66	2.54
TA22	31,587	32,401	2.58	2.71	31,807	0.70	0.56	32,734	3.63	3.59	32,370	2.48	2.52	32,264	2.14	2.28
TA23	33,920	34,820	2.65	2.62	34,064	0.42	0.35	35,038	3.30	3.40	34,788	2.56	2.53	34,635	2.11	2.18
TA24	31,661	32,393	2.31	2.12	31,841	0.57	0.48	32,802	3.60	3.39	32,445	2.48	2.41	32,333	2.12	2.06
TA25	34,557	35,387	2.40	2.42	34,706	0.43	0.41	35,814	3.64	3.69	35,482	2.68	2.81	35,293	2.13	2.11
TA26	32,564	33,459	2.75	2.84	32,802	0.73	0.21	33,828	3.88	4.16	33,489	2.84	3.12	33,275	2.18	2.11

Instance	BKS	Tuned-NLS (1-hour)			Random-NLS (1-hour)			Race-NLS (1-hour)			Race-NLS (2-hours)			Race-NLS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
TA27	32,922	33,796	2.65	2.63	33,268	1.05	0.98	34,073	3.50	3.53	33,914	3.01	3.17	33,703	2.37	2.32
TA28	32,412	33,314	2.78	2.76	32,626	0.66	0.56	33,574	3.59	3.77	33,341	2.87	2.72	33,119	2.18	2.08
TA29	33,600	34,450	2.53	2.58	33,790	0.57	0.52	34,697	3.27	3.25	34,527	2.76	3.01	34,340	2.20	2.15
TA30	32,262	33,193	2.88	2.94	32,433	0.53	0.33	33,642	4.28	4.46	33,257	3.08	3.17	32,959	2.16	2.01
TA31	64,802	67,900	4.78	4.75	66,220	2.19	2.21	72,789	12.32	12.67	71,503	10.34	10.34	70,541	8.86	9.07
TA32	68,051	71,509	5.08	4.99	69,910	2.73	2.71	76,998	13.15	12.97	75,396	10.79	10.92	74,371	9.29	8.92
TA33	63,162	66,633	5.49	5.68	65,128	3.11	3.18	71,800	13.68	14.60	70,573	11.73	11.79	69,510	10.05	9.89
TA34	68,226	71,920	5.41	5.19	70,309	3.05	3.05	76,811	12.58	12.95	75,305	10.38	10.60	74,412	9.07	9.32
TA35	69,351	72,671	4.79	4.69	70,981	2.35	2.33	77,073	11.13	11.36	75,902	9.45	9.79	75,045	8.21	8.05
TA36	66,841	69,916	4.60	4.57	68,690	2.77	2.56	75,112	12.37	11.88	74,016	10.73	11.12	72,786	8.89	8.78
TA37	66,253	69,320	4.63	4.71	67,889	2.47	2.42	74,268	12.10	12.20	72,750	9.81	9.74	72,050	8.75	8.73
TA38	64,332	67,601	5.08	4.94	66,135	2.80	2.62	72,935	13.37	13.39	71,038	10.42	9.49	70,295	9.27	9.30
TA39	62,981	66,015	4.82	4.93	64,641	2.64	2.69	70,706	12.27	12.07	69,750	10.75	10.59	68,480	8.73	8.79
TA40	68,770	72,101	4.84	4.64	70,781	2.92	2.95	76,878	11.79	12.08	75,747	10.15	10.09	74,851	8.84	8.69
TA41	87,114	92,798	6.53	6.34	90,594	3.99	3.97	98,952	13.59	14.10	97,651	12.10	12.48	96,585	10.87	11.02
TA42	82,820	87,989	6.24	6.21	85,765	3.56	3.52	93,812	13.27	13.53	92,423	11.60	11.58	91,645	10.66	10.94
TA43	79,931	85,926	7.50	7.35	82,741	3.52	3.27	92,658	15.92	15.75	91,038	13.90	14.37	89,136	11.52	11.96
TA44	86,446	91,570	5.93	5.88	89,254	3.25	3.10	97,557	12.85	13.02	96,330	11.43	11.22	94,761	9.62	9.15
TA45	86,377	91,696	6.16	5.96	89,619	3.75	3.79	97,477	12.85	13.08	95,824	10.94	11.22	94,600	9.52	9.74
TA46	86,587	91,625	5.82	5.76	89,299	3.13	3.01	97,188	12.24	12.34	95,710	10.54	11.00	94,997	9.71	9.88
TA47	88,750	93,136	4.94	4.80	91,607	3.22	3.11	98,690	11.20	11.44	97,759	10.15	10.63	96,414	8.64	9.05
TA48	86,727	92,063	6.15	5.96	89,465	3.16	3.20	97,441	12.35	12.54	96,149	10.86	10.70	95,135	9.69	9.59
TA49	85,441	90,493	5.91	5.75	88,280	3.32	3.21	96,431	12.86	13.47	95,223	11.45	11.50	93,584	9.53	9.59
TA50	87,998	93,304	6.03	5.80	91,210	3.65	3.55	99,461	13.03	13.33	97,857	11.20	11.49	96,753	9.95	9.84
TA51	125,831	132,750	5.50	5.37	129,807	3.16	3.17	140,128	11.36	11.47	138,384	9.98	9.95	136,936	8.83	9.13
TA52	119,247	125,715	5.42	5.29	123,080	3.21	3.15	133,198	11.70	11.73	130,723	9.62	10.20	129,590	8.67	8.60
TA53	116,459	124,050	6.52	6.59	120,739	3.68	3.76	132,100	13.43	13.20	129,185	10.93	11.35	128,143	10.03	10.30
TA54	120,261	126,974	5.58	5.45	124,560	3.57	3.56	133,697	11.17	11.30	132,123	9.86	10.24	130,627	8.62	8.71
TA55	118,184	124,384	5.25	5.09	121,934	3.17	3.28	132,651	12.24	12.38	130,199	10.17	10.32	128,798	8.98	9.18
TA56	120,586	127,168	5.46	5.03	124,121	2.93	2.80	133,780	10.94	11.16	132,424	9.82	10.15	130,858	8.52	8.76
TA57	122,880	129,412	5.32	5.34	126,568	3.00	3.10	135,944	10.63	10.29	134,098	9.13	9.14	133,400	8.56	8.90
TA58	122,489	129,219	5.49	5.39	126,424	3.21	3.23	135,733	10.81	11.14	134,483	9.79	9.95	132,946	8.54	8.98

Instance	BKS	Tuned-NLS (1-hour)			Random-NLS (1-hour)			Race-NLS (1-hour)			Race-NLS (2-hours)			Race-NLS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
TA59	121,872	128,032	5.05	5.15	125,321	2.83	2.91	134,803	10.61	10.61	133,344	9.41	9.72	132,255	8.52	9.10
TA60	123,954	130,151	5.00	4.95	127,836	3.13	3.07	136,312	9.97	9.90	135,345	9.19	9.36	133,686	7.85	8.25
TA61	253,167	270,416	6.81	6.91	261,736	3.38	3.35	289,749	14.45	15.03	285,722	12.86	13.36	282,333	11.52	11.46
TA62	241,925	259,219	7.15	7.51	250,674	3.62	3.61	278,834	15.26	15.66	275,290	13.79	13.89	271,864	12.38	12.30
TA63	237,832	254,090	6.84	6.86	245,244	3.12	3.14	274,112	15.25	15.13	268,813	13.03	13.41	265,690	11.71	11.40
TA64	227,522	241,524	6.15	6.70	234,547	3.09	3.14	260,823	14.64	14.54	257,686	13.26	13.74	254,172	11.71	12.08
TA65	240,301	254,385	5.86	6.25	247,217	2.88	2.80	274,977	14.43	14.33	271,051	12.80	12.95	267,645	11.38	11.06
TA66	232,342	247,827	6.66	7.04	239,991	3.29	3.25	268,770	15.68	16.19	265,060	14.08	13.58	261,805	12.68	13.09
TA67	240,366	254,896	6.04	6.08	247,858	3.12	3.04	275,578	14.65	14.95	271,082	12.78	13.74	268,319	11.63	11.84
TA68	230,945	248,830	7.74	7.59	239,545	3.72	3.71	270,320	17.05	17.53	265,343	14.89	14.80	262,624	13.72	13.84
TA69	247,526	265,072	7.09	6.94	256,184	3.50	3.50	286,389	15.70	15.87	282,886	14.29	14.16	279,165	12.78	13.28
TA70	242,933	259,644	6.88	6.78	250,310	3.04	2.97	280,610	15.51	15.43	276,118	13.66	13.95	272,209	12.05	12.38
TA71	298,385	321,343	7.69	7.26	310,248	3.98	3.93	344,709	15.52	15.87	340,001	13.95	13.92	337,748	13.19	13.09
TA72	273,674	296,746	8.43	8.32	285,330	4.26	4.29	321,347	17.42	17.47	316,936	15.81	16.08	313,139	14.42	14.47
TA73	288,114	309,880	7.55	7.29	299,283	3.88	3.93	333,328	15.69	15.49	328,923	14.16	14.16	326,667	13.38	13.35
TA74	301,044	325,976	8.28	7.91	313,721	4.21	4.35	348,604	15.80	15.73	344,505	14.44	14.51	339,493	12.77	12.72
TA75	284,233	306,914	7.98	8.27	295,588	3.99	4.03	332,556	17.00	17.39	327,709	15.30	15.58	324,425	14.14	14.18
TA76	269,686	293,274	8.75	8.99	281,404	4.34	4.27	317,154	17.60	18.03	313,615	16.29	16.68	308,993	14.58	14.38
TA77	279,463	298,576	6.84	6.93	290,439	3.93	3.96	327,670	17.25	17.06	322,982	15.57	15.72	317,705	13.68	13.74
TA78	290,908	312,906	7.56	7.39	302,483	3.98	3.94	336,366	15.63	15.97	333,353	14.59	14.69	329,696	13.33	13.34
TA79	301,970	322,091	6.66	6.24	313,798	3.92	3.87	346,126	14.62	14.60	343,292	13.68	13.83	339,602	12.46	12.89
TA80	291,283	317,790	9.10	9.37	304,361	4.49	4.50	343,218	17.83	18.33	338,218	16.11	16.28	334,375	14.79	15.21
TA81	365,463	392,704	7.45	7.16	381,157	4.29	4.26	421,122	15.23	15.43	416,594	13.99	14.04	413,280	13.08	13.64
TA82	372,449	399,257	7.20	7.15	387,436	4.02	4.07	427,229	14.71	14.78	422,502	13.44	13.26	418,917	12.48	12.34
TA83	370,027	394,618	6.65	6.28	384,158	3.82	3.73	423,758	14.52	14.47	419,365	13.33	13.51	413,866	11.85	12.30
TA84	372,393	401,795	7.90	7.15	388,514	4.33	4.37	427,678	14.85	15.03	422,992	13.59	13.96	419,302	12.60	12.56
TA85	368,915	396,877	7.58	7.62	383,798	4.03	4.10	426,429	15.59	15.90	419,414	13.69	13.95	416,138	12.80	12.87
TA86	370,908	400,158	7.89	7.46	387,785	4.55	4.58	426,910	15.10	15.59	423,626	14.21	14.50	420,111	13.27	13.52
TA87	373,408	401,367	7.49	7.64	389,548	4.32	4.40	430,105	15.18	15.34	426,183	14.13	14.26	421,241	12.81	12.71
TA88	384,525	410,725	6.81	7.02	399,672	3.94	3.83	437,438	13.76	14.15	431,939	12.33	12.54	428,979	11.56	12.10
TA89	374,423	401,802	7.31	6.91	389,849	4.12	4.17	428,293	14.39	14.56	424,068	13.26	13.32	419,902	12.15	12.35
TA90	379,296	406,143	7.08	6.75	394,158	3.92	3.87	434,406	14.53	14.87	430,659	13.54	13.72	428,372	12.94	13.01

Instance	BKS	Tuned-NLS (1-hour)			Random-NLS (1-hour)			Race-NLS (1-hour)			Race-NLS (2-hours)			Race-NLS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
TA91	1,041,023	1,148,508	10.32	10.43	1,085,215	4.25	4.19	1,075,667	3.40	3.58	1,059,845	2.27	2.35	1,050,541	1.82	1.55
TA92	1,028,828	1,145,708	11.36	11.35	1,078,381	4.82	4.81	1,041,454	2.00	1.94	1,024,562	1.73	1.76	1,011,652	2.10	1.79
TA93	1,042,357	1,153,140	10.63	10.60	1,086,885	4.27	4.37	1,078,428	3.57	3.57	1,060,945	2.44	2.16	1,051,038	1.57	1.34
TA94	1,025,564	1,131,378	10.32	10.25	1,067,340	4.07	3.98	1,047,687	2.67	2.73	1,034,789	2.18	2.07	1,017,750	1.79	1.55
TA95	1,028,963	1,143,275	11.11	11.09	1,074,070	4.38	4.32	1,064,063	3.42	3.29	1,048,655	2.51	2.26	1,037,184	1.68	1.40
TA96	998,340	1,123,266	12.51	12.82	1,051,119	5.29	5.26	1,028,056	3.09	3.03	1,014,386	2.10	1.68	1,012,791	2.61	1.52
TA97	1,042,570	1,176,341	12.83	12.79	1,099,889	5.50	5.50	1,074,185	3.21	3.42	1,067,113	2.99	2.23	1,051,118	1.48	1.30
TA98	1,035,915	1,155,045	11.50	11.57	1,084,628	4.70	4.68	1,064,682	3.21	2.52	1,041,440	1.81	1.73	1,028,731	1.64	1.44
TA99	1,015,280	1,136,400	11.93	11.99	1,065,532	4.95	4.89	1,055,765	3.99	3.92	1,040,956	2.73	2.67	1,032,082	2.15	1.89
TA100	1,021,865	1,146,463	12.19	12.00	1,074,044	5.11	4.97	1,047,289	2.67	2.27	1,032,929	2.00	2.09	1,019,728	1.56	1.35
TA101	1,219,341	1,360,723	11.59	11.65	1,277,518	4.77	4.75	1,057,469	13.28	12.92	1,043,230	14.44	14.21	1,032,311	15.34	15.22
TA102	1,233,161	1,373,639	11.39	11.70	1,296,497	5.14	5.08	1,089,780	11.63	11.51	1,077,880	12.59	12.36	1,064,058	13.71	13.97
TA103	1,259,605	1,403,593	11.43	11.23	1,317,405	4.59	4.67	1,073,707	14.76	14.74	1,059,625	15.88	15.75	1,045,799	16.97	17.04
TA104	1,228,027	1,380,535	12.42	12.50	1,294,611	5.42	5.30	1,035,831	15.65	15.80	1,019,482	16.98	16.67	1,009,579	17.79	17.60
TA105	1,215,854	1,358,560	11.74	11.77	1,278,039	5.11	5.06	1,046,903	13.90	13.93	1,031,472	15.16	14.53	1,018,692	16.22	16.31
TA106	1,218,757	1,359,250	11.53	11.73	1,278,011	4.86	4.83	1,067,792	12.39	12.01	1,046,539	14.13	13.89	1,034,571	15.11	15.25
TA107	1,234,330	1,379,611	11.77	11.77	1,300,229	5.34	5.36	1,038,328	15.88	15.70	1,023,475	17.08	17.43	1,008,828	18.27	18.25
TA108	1,240,105	1,378,917	11.19	11.15	1,295,697	4.48	4.52	1,067,096	13.95	13.76	1,050,561	15.28	14.99	1,039,850	16.15	16.24
TA109	1,220,058	1,366,099	11.97	12.00	1,286,509	5.45	5.41	1,063,882	12.80	12.32	1,064,979	13.98	13.57	1,035,538	15.12	15.11
TA110	1,235,113	1,384,859	12.12	12.02	1,300,030	5.26	5.27	1,085,528	12.11	11.96	1,070,584	13.32	12.94	1,056,506	14.46	14.44
TA111	6,558,109	7,519,528	14.66	14.66	7,011,307	6.91	6.81	6,438,430	1.82	1.53	6,382,729	2.67	2.17	6,324,674	3.56	3.63
TA112	6,679,339	7,631,097	14.25	14.33	7,111,080	6.46	6.45	6,516,774	2.43	2.25	6,450,907	3.42	3.13	6,382,589	4.44	4.20
TA113	6,624,644	7,571,486	14.29	14.31	7,053,161	6.47	6.49	6,499,056	1.90	1.84	6,423,485	3.04	2.62	6,357,917	4.03	3.91
TA114	6,646,006	7,597,948	14.32	14.37	7,089,213	6.67	6.72	6,480,144	2.83	2.66	6,378,075	4.03	3.71	6,313,089	5.01	4.90
TA115	6,587,110	7,553,325	14.67	14.51	7,039,114	6.86	6.84	6,474,560	1.75	1.48	6,404,200	2.78	2.29	6,362,574	3.41	2.98
TA116	6,603,291	7,590,925	14.96	14.95	7,061,402	6.94	6.81	6,440,189	2.47	2.07	6,358,034	3.71	3.33	6,290,421	4.74	4.86
TA117	6,602,685	7,520,466	13.90	13.83	7,006,968	6.12	6.11	6,386,523	3.27	3.11	6,311,057	4.42	4.12	6,253,237	5.29	5.30
TA118	6,629,065	7,595,907	14.58	14.48	7,081,361	6.82	6.83	6,488,142	2.13	1.98	6,410,902	3.29	2.88	6,338,346	4.39	4.37
TA119	6,587,638	7,520,394	14.16	14.26	7,013,790	6.47	6.37	6,453,000	2.49	2.39	6,363,739	3.40	3.34	6,308,063	4.24	4.00
TA120	6,623,849	7,583,513	14.49	14.46	7,063,029	6.63	6.61	6,426,544	3.42	3.29	6,343,023	4.24	3.84	6,291,780	5.01	5.14

Table B.2: The table presents the complete results for the FSP with Local Search. Each row represents a problem instance, and "BKS" stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature.

Instance	BKS	Tuned-LS (1-hour)			Random-LS (1-hour)			Race-LS (1-hour)			Race-LS (2-hours)			Race-LS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
TA1	14,033	14,033	0.00	0.00	14,033	0.00	0.00	14,033	0.00	0.00	14,033	0.00	0.00	14,033	0.00	0.00
TA2	15,151	15,151	0.00	0.00	15,151	0.00	0.00	15,152	0.01	0.00	15,151	0.00	0.00	15,151	0.00	0.00
TA3	13,301	13,304	0.02	0.00	13,301	0.00	0.00	13,301	0.00	0.00	13,301	0.00	0.00	13,301	0.00	0.00
TA4	15,447	15,447	0.00	0.00	15,447	0.00	0.00	15,447	0.00	0.00	15,447	0.00	0.00	15,447	0.00	0.00
TA5	13,529	13,529	0.00	0.00	13,529	0.00	0.00	13,529	0.00	0.00	13,529	0.00	0.00	13,529	0.00	0.00
TA6	13,123	13,123	0.00	0.00	13,123	0.00	0.00	13,123	0.00	0.00	13,123	0.00	0.00	13,123	0.00	0.00
TA7	13,548	13,548	0.00	0.00	13,549	0.01	0.00	13,549	0.01	0.00	13,548	0.00	0.00	13,548	0.00	0.00
TA8	13,948	13,948	0.00	0.00	13,948	0.00	0.00	13,948	0.00	0.00	13,948	0.00	0.00	13,948	0.00	0.00
TA9	14,295	14,295	0.00	0.00	14,298	0.02	0.00	14,318	0.16	0.15	14,313	0.12	0.15	14,309	0.10	0.14
TA10	12,943	12,943	0.00	0.00	12,943	0.00	0.00	12,943	0.00	0.00	12,943	0.00	0.00	12,943	0.00	0.00
TA11	20,911	20,911	0.00	0.00	20,911	0.00	0.00	20,911	0.00	0.00	20,911	0.00	0.00	20,911	0.00	0.00
TA12	22,440	22,440	0.00	0.00	22,440	0.00	0.00	22,440	0.00	0.00	22,440	0.00	0.00	22,440	0.00	0.00
TA13	19,833	19,833	0.00	0.00	19,833	0.00	0.00	19,833	0.00	0.00	19,833	0.00	0.00	19,833	0.00	0.00
TA14	18,710	18,710	0.00	0.00	18,711	0.00	0.00	18,714	0.02	0.00	18,711	0.01	0.00	18,710	0.00	0.00
TA15	18,641	18,641	0.00	0.00	18,641	0.00	0.00	18,641	0.00	0.00	18,641	0.00	0.00	18,641	0.00	0.00
TA16	19,245	19,245	0.00	0.00	19,245	0.00	0.00	19,245	0.00	0.00	19,245	0.00	0.00	19,245	0.00	0.00
TA17	18,363	18,363	0.00	0.00	18,363	0.00	0.00	18,364	0.00	0.00	18,363	0.00	0.00	18,363	0.00	0.00
TA18	20,241	20,241	0.00	0.00	20,241	0.00	0.00	20,241	0.00	0.00	20,241	0.00	0.00	20,241	0.00	0.00
TA19	20,330	20,330	0.00	0.00	20,330	0.00	0.00	20,330	0.00	0.00	20,330	0.00	0.00	20,330	0.00	0.00
TA20	21,320	21,320	0.00	0.00	21,320	0.00	0.00	21,320	0.00	0.00	21,320	0.00	0.00	21,320	0.00	0.00
TA21	33,623	33,623	0.00	0.00	33,623	0.00	0.00	33,623	0.00	0.00	33,623	0.00	0.00	33,623	0.00	0.00
TA22	31,587	31,587	0.00	0.00	31,587	0.00	0.00	31,588	0.00	0.00	31,587	0.00	0.00	31,587	0.00	0.00
TA23	33,920	33,920	0.00	0.00	33,920	0.00	0.00	33,920	0.00	0.00	33,920	0.00	0.00	33,920	0.00	0.00
TA24	31,661	31,661	0.00	0.00	31,661	0.00	0.00	31,663	0.01	0.00	31,661	0.00	0.00	31,661	0.00	0.00
TA25	34,557	34,557	0.00	0.00	34,559	0.01	0.00	34,560	0.01	0.00	34,559	0.00	0.00	34,557	0.00	0.00
TA26	32,564	32,564	0.00	0.00	32,564	0.00	0.00	32,564	0.00	0.00	32,564	0.00	0.00	32,564	0.00	0.00

Instance	BKS	Tuned-LS (1-hour)			Random-LS (1-hour)			Race-LS (1-hour)			Race-LS (2-hours)			Race-LS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
TA27	32,922	32,922	0.00	0.00	32,922	0.00	0.00	32,922	0.00	0.00	32,922	0.00	0.00	32,922	0.00	0.00
TA28	32,412	32,414	0.00	0.00	32,412	0.00	0.00	32,412	0.00	0.00	32,412	0.00	0.00	32,412	0.00	0.00
TA29	33,600	33,600	0.00	0.00	33,600	0.00	0.00	33,600	0.00	0.00	33,600	0.00	0.00	33,600	0.00	0.00
TA30	32,262	32,265	0.01	0.00	32,268	0.02	0.01	32,277	0.05	0.03	32,271	0.03	0.03	32,266	0.01	0.00
TA31	64,802	65,567	1.18	1.21	65,911	1.71	1.75	66,404	2.47	2.49	66,313	2.33	2.43	66,144	2.07	2.05
TA32	68,051	69,131	1.59	1.61	69,412	2.00	2.04	70,016	2.89	2.94	69,909	2.73	2.80	69,731	2.47	2.38
TA33	63,162	64,214	1.67	1.71	64,528	2.16	2.16	65,240	3.29	3.31	65,082	3.04	3.02	64,909	2.77	2.79
TA34	68,226	69,273	1.54	1.56	69,712	2.18	2.15	70,251	2.97	2.96	70,178	2.86	2.88	69,969	2.56	2.54
TA35	69,351	70,196	1.22	1.24	70,603	1.81	1.82	71,147	2.59	2.71	71,103	2.53	2.53	70,860	2.18	2.18
TA36	66,841	67,827	1.48	1.48	68,053	1.81	1.80	68,435	2.38	2.36	68,321	2.21	2.24	68,202	2.04	2.05
TA37	66,253	67,074	1.24	1.28	67,414	1.75	1.74	67,898	2.48	2.47	67,818	2.36	2.38	67,718	2.21	2.28
TA38	64,332	65,248	1.42	1.42	65,534	1.87	1.92	66,079	2.72	2.79	65,953	2.52	2.56	65,849	2.36	2.36
TA39	62,981	63,759	1.24	1.25	64,198	1.93	1.93	64,671	2.68	2.73	64,613	2.59	2.61	64,429	2.30	2.28
TA40	68,770	69,871	1.60	1.60	70,132	1.98	2.03	70,774	2.91	2.97	70,664	2.75	2.75	70,480	2.49	2.54
TA41	87,114	89,379	2.60	2.66	89,781	3.06	3.11	90,410	3.78	3.89	90,384	3.75	3.83	90,092	3.42	3.45
TA42	82,820	85,080	2.73	2.75	85,584	3.34	3.33	85,863	3.67	3.72	85,813	3.61	3.64	85,565	3.31	3.37
TA43	79,931	81,901	2.46	2.49	82,301	2.96	2.95	82,891	3.70	3.81	82,695	3.46	3.47	82,410	3.10	3.10
TA44	86,446	88,390	2.25	2.27	88,666	2.57	2.57	89,500	3.53	3.48	89,215	3.20	3.26	89,200	3.19	3.12
TA45	86,377	88,621	2.60	2.64	88,777	2.78	2.81	89,691	3.84	3.79	89,363	3.46	3.48	89,177	3.24	3.32
TA46	86,587	88,499	2.21	2.22	89,008	2.80	2.82	89,534	3.40	3.43	89,332	3.17	3.19	89,163	2.97	2.98
TA47	88,750	90,599	2.08	2.10	90,930	2.46	2.57	91,005	2.54	2.55	90,922	2.45	2.46	90,758	2.26	2.33
TA48	86,727	88,718	2.30	2.32	88,988	2.61	2.62	89,604	3.32	3.37	89,479	3.17	3.12	89,274	2.94	2.99
TA49	85,441	87,446	2.35	2.37	87,840	2.81	2.84	88,351	3.41	3.39	88,186	3.21	3.28	87,961	2.95	2.99
TA50	87,998	90,198	2.50	2.53	90,426	2.76	2.74	90,802	3.19	3.25	90,642	3.00	3.01	90,480	2.82	2.83
TA51	125,831	128,869	2.41	2.46	129,627	3.02	3.02	130,315	3.56	3.61	129,936	3.26	3.34	129,783	3.14	3.18
TA52	119,247	122,375	2.62	2.68	123,153	3.28	3.25	123,708	3.74	3.79	123,542	3.60	3.63	123,219	3.33	3.36
TA53	116,459	119,650	2.74	2.79	120,286	3.29	3.21	121,462	4.30	4.34	120,860	3.78	3.84	120,661	3.61	3.69
TA54	120,261	123,665	2.83	2.85	124,569	3.58	3.59	124,755	3.74	3.74	124,569	3.58	3.63	124,384	3.43	3.47
TA55	118,184	121,220	2.57	2.64	121,750	3.02	3.01	122,402	3.57	3.61	122,150	3.36	3.38	121,830	3.09	3.11
TA56	120,586	123,448	2.37	2.39	124,104	2.92	2.99	124,638	3.36	3.50	124,341	3.11	3.21	124,076	2.89	2.93
TA57	122,880	125,951	2.50	2.51	126,487	2.94	2.97	126,858	3.24	3.26	126,744	3.14	3.16	126,410	2.87	2.90
TA58	122,489	125,675	2.60	2.64	126,165	3.00	3.03	126,790	3.51	3.56	126,482	3.26	3.38	126,362	3.16	3.17

Instance	BKS	Tuned-LS (1-hour)			Random-LS (1-hour)			Race-LS (1-hour)			Race-LS (2-hours)			Race-LS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
TA59	121,872	124,818	2.42	2.45	125,577	3.04	3.04	126,038	3.42	3.50	125,672	3.12	3.07	125,605	3.06	3.07
TA60	123,954	126,689	2.21	2.18	127,316	2.71	2.73	127,826	3.12	3.12	127,539	2.89	2.92	127,371	2.76	2.81
TA61	253,167	259,037	2.32	2.32	264,848	4.61	4.66	265,773	4.98	5.04	265,571	4.90	4.88	264,816	4.60	4.61
TA62	241,925	248,302	2.64	2.66	255,644	5.67	5.74	256,386	5.98	5.97	255,809	5.74	5.74	254,997	5.40	5.46
TA63	237,832	243,073	2.20	2.21	249,837	5.05	5.11	250,689	5.41	5.47	250,428	5.30	5.30	249,559	4.93	4.94
TA64	227,522	232,705	2.28	2.30	240,427	5.67	5.68	241,188	6.01	6.05	240,526	5.72	5.82	239,971	5.47	5.46
TA65	240,301	245,402	2.12	2.15	252,042	4.89	4.94	252,552	5.10	5.20	252,200	4.95	4.97	252,012	4.87	4.93
TA66	232,342	238,517	2.66	2.68	244,500	5.23	5.30	245,303	5.58	5.65	244,743	5.34	5.40	244,299	5.15	5.20
TA67	240,366	245,451	2.12	2.12	252,921	5.22	5.24	254,011	5.68	5.84	253,411	5.43	5.51	252,884	5.21	5.29
TA68	230,945	237,044	2.64	2.70	245,775	6.42	6.38	247,017	6.96	6.94	246,316	6.66	6.71	245,981	6.51	6.44
TA69	247,526	254,466	2.80	2.81	260,154	5.10	5.10	260,651	5.30	5.34	260,354	5.18	5.22	259,920	5.01	5.01
TA70	242,933	248,305	2.21	2.23	254,846	4.90	4.94	255,805	5.30	5.31	255,353	5.11	5.20	255,037	4.98	4.98
TA71	298,385	309,525	3.73	3.76	316,173	5.96	5.91	316,415	6.04	6.14	315,775	5.83	5.80	314,900	5.53	5.54
TA72	273,674	285,771	4.42	4.41	293,171	7.12	7.12	293,260	7.16	7.14	292,771	6.98	6.99	291,838	6.64	6.67
TA73	288,114	298,875	3.73	3.74	307,748	6.81	6.87	308,182	6.97	6.96	307,456	6.71	6.76	307,018	6.56	6.58
TA74	301,044	312,717	3.88	3.89	319,214	6.04	6.07	319,842	6.24	6.30	319,287	6.06	6.10	318,294	5.73	5.80
TA75	284,233	295,457	3.95	3.95	303,776	6.88	6.90	304,538	7.14	7.13	304,273	7.05	7.08	303,196	6.67	6.83
TA76	269,686	281,169	4.26	4.30	288,336	6.92	6.92	289,349	7.29	7.28	288,526	6.99	7.04	288,209	6.87	6.85
TA77	279,463	290,161	3.83	3.83	296,190	5.99	5.99	296,700	6.17	6.14	296,318	6.03	6.12	295,646	5.79	5.76
TA78	290,908	301,905	3.78	3.79	308,372	6.00	6.00	308,971	6.21	6.23	308,262	5.97	6.00	307,316	5.64	5.68
TA79	301,970	312,992	3.65	3.67	318,509	5.48	5.43	319,505	5.81	5.95	318,719	5.55	5.57	317,841	5.26	5.33
TA80	291,283	303,728	4.27	4.28	312,189	7.18	7.29	313,275	7.55	7.63	312,502	7.28	7.27	311,719	7.02	6.99
TA81	365,463	381,444	4.37	4.40	389,051	6.45	6.48	389,260	6.51	6.61	388,150	6.21	6.20	387,554	6.04	6.02
TA82	372,449	387,520	4.05	4.06	395,528	6.20	6.16	396,291	6.40	6.45	395,717	6.25	6.29	394,620	5.95	6.05
TA83	370,027	384,417	3.89	3.89	392,468	6.06	6.09	391,908	5.91	5.92	391,575	5.82	5.83	390,622	5.57	5.60
TA84	372,393	388,193	4.24	4.27	394,917	6.05	6.07	395,246	6.14	6.16	394,709	5.99	6.06	393,704	5.72	5.73
TA85	368,915	384,049	4.10	4.11	391,696	6.18	6.24	391,632	6.16	6.18	390,917	5.96	5.95	390,114	5.75	5.66
TA86	370,908	386,840	4.30	4.32	395,643	6.67	6.74	396,554	6.91	6.94	395,314	6.58	6.57	394,451	6.35	6.45
TA87	373,408	389,114	4.21	4.29	397,695	6.50	6.51	397,993	6.58	6.67	397,066	6.34	6.38	395,828	6.00	6.17
TA88	384,525	398,925	3.74	3.79	406,125	5.62	5.62	406,611	5.74	5.72	405,682	5.50	5.46	404,611	5.22	5.32
TA89	374,423	389,604	4.05	4.09	396,531	5.90	5.93	397,197	6.08	6.07	396,262	5.83	5.91	395,628	5.66	5.73
TA90	379,296	394,252	3.94	3.98	400,747	5.66	5.64	400,645	5.63	5.65	399,545	5.34	5.41	398,950	5.18	5.22

Instance	BKS	Tuned-LS (1-hour)			Random-LS (1-hour)			Race-LS (1-hour)			Race-LS (2-hours)			Race-LS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
TA91	1,041,023	1,087,312	4.45	4.43	1,120,820	7.67	7.77	986,834	5.21	5.19	984,738	5.41	5.38	983,502	5.53	5.52
TA92	1,028,828	1,082,015	5.17	5.17	1,119,067	8.77	8.88	958,885	6.80	6.79	954,970	7.18	7.11	953,304	7.34	7.38
TA93	1,042,357	1,092,079	4.77	4.81	1,130,372	8.44	8.48	993,914	4.65	4.58	991,186	4.91	4.95	988,882	5.13	5.16
TA94	1,025,564	1,077,808	5.09	5.09	1,116,045	8.82	8.88	966,101	5.80	5.76	963,617	6.04	6.01	962,168	6.18	6.15
TA95	1,028,963	1,077,576	4.72	4.75	1,114,743	8.34	8.37	980,198	4.74	4.75	977,059	5.04	4.89	976,208	5.13	5.14
TA96	998,340	1,053,569	5.53	5.54	1,093,676	9.55	9.65	947,781	5.06	5.00	945,839	5.26	5.18	942,641	5.58	5.56
TA97	1,042,570	1,105,260	6.01	6.02	1,139,943	9.34	9.32	994,887	4.57	4.53	992,389	4.81	4.80	990,507	4.99	4.97
TA98	1,035,915	1,090,985	5.32	5.39	1,128,131	8.90	9.05	974,633	5.92	5.87	971,031	6.26	6.21	969,031	6.46	6.42
TA99	1,015,280	1,070,119	5.40	5.40	1,108,351	9.17	9.24	973,496	4.12	4.02	973,395	4.13	4.06	974,625	4.54	4.36
TA100	1,021,865	1,080,383	5.73	5.76	1,117,652	9.37	9.37	960,196	6.03	6.02	958,350	6.22	6.21	956,730	6.37	6.36
TA101	1,219,341	1,281,640	5.11	5.14	1,323,863	8.57	8.65	980,676	19.57	19.54	979,253	19.69	19.60	976,130	19.95	19.94
TA102	1,233,161	1,301,582	5.55	5.58	1,343,285	8.93	8.93	1,007,988	18.26	18.24	1,006,338	18.39	18.34	1,003,956	18.59	18.55
TA103	1,259,605	1,317,961	4.63	4.63	1,356,001	7.65	7.69	990,848	21.34	21.41	989,164	21.47	21.43	999,313	21.14	21.57
TA104	1,228,027	1,297,884	5.69	5.69	1,337,986	8.95	9.11	952,364	22.45	22.42	951,778	22.50	22.53	948,356	22.77	22.73
TA105	1,215,854	1,282,419	5.47	5.44	1,333,646	9.69	9.79	956,035	21.37	21.36	952,446	21.66	21.64	964,550	21.27	21.67
TA106	1,218,757	1,285,605	5.48	5.55	1,327,531	8.93	8.94	978,347	19.73	19.70	975,170	19.99	19.92	984,368	19.80	20.13
TA107	1,234,330	1,303,529	5.61	5.61	1,346,294	9.07	9.11	949,178	23.10	23.01	947,520	23.24	23.19	946,329	23.33	23.34
TA108	1,240,105	1,298,635	4.72	4.72	1,340,767	8.12	8.22	978,499	21.10	21.14	989,050	20.78	21.14	975,856	21.31	21.31
TA109	1,220,058	1,291,546	5.86	5.87	1,331,436	9.13	9.09	991,536	19.34	19.65	978,801	19.77	19.73	975,617	20.04	19.99
TA110	1,235,113	1,303,637	5.55	5.55	1,343,359	8.76	8.70	1,001,478	18.92	18.87	997,818	19.21	19.20	996,118	19.35	19.33
TA111	6,558,109	7,035,306	7.28	7.28	7,283,677	11.06	11.10	6,002,478	8.47	8.44	5,993,702	8.61	8.51	5,977,910	8.85	8.79
TA112	6,679,339	7,142,075	6.93	6.85	7,368,259	10.31	10.34	6,022,445	9.83	9.74	6,001,608	10.15	10.11	6,010,430	10.01	10.30
TA113	6,624,644	7,086,850	6.98	6.92	7,312,899	10.39	10.38	6,008,190	9.31	9.27	5,999,112	9.44	9.38	5,984,416	9.66	9.64
TA114	6,646,006	7,094,979	6.76	6.71	7,346,946	10.55	10.51	5,953,401	10.42	10.49	5,944,149	10.56	10.66	5,930,905	10.76	10.73
TA115	6,587,110	7,082,172	7.52	7.55	7,316,324	11.07	11.06	6,046,537	8.21	8.49	6,030,532	8.45	8.62	6,000,234	8.91	8.91
TA116	6,603,291	7,107,530	7.64	7.65	7,368,827	11.59	11.65	5,978,462	9.46	9.51	5,973,515	9.54	9.56	5,950,002	9.89	9.92
TA117	6,602,685	7,045,416	6.71	6.72	7,302,968	10.61	10.66	5,905,679	10.56	10.50	5,894,538	10.73	10.74	5,876,773	10.99	10.93
TA118	6,629,065	7,117,959	7.38	7.40	7,356,109	10.97	11.16	5,999,655	9.49	9.40	5,989,411	9.65	9.70	5,972,479	9.90	9.91
TA119	6,587,638	7,043,724	6.92	6.99	7,269,865	10.36	10.34	5,993,699	9.02	8.97	5,986,388	9.13	9.12	5,970,447	9.37	9.35
TA120	6,623,849	7,090,323	7.04	7.02	7,322,877	10.55	10.60	5,949,613	10.18	10.09	5,941,632	10.30	10.32	5,924,859	10.55	10.55

C
Complete Results for the TSP

Table C.1: The table presents the complete results for the TSP without Local Search. Each row represents a problem instance, and “BKS” stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature.

Instance	BKS	Tuned-NLS (1-hour)			Random-NLS (1-hour)			Race-NLS (1-hour)			Race-NLS (2-hours)			Race-NLS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
a280	2,579	28,911	1,021.01	1,023.17	9,809	280.34	273.13	28,851	1,018.69	1,021.60	28,652	1,010.97	1,010.90	28,372	1,000.13	1,007.50
ali535	202,339	3,159,375	1,461.43	1,462.66	1,330,862	557.74	549.90	3,157,891	1,460.69	1,463.14	3,147,990	1,455.80	1,458.50	3,139,517	1,451.61	1,452.85
att48	10,628	31,219	193.74	194.30	11,994	12.86	9.89	29,369	176.34	191.71	27,447	158.25	187.51	24,221	127.90	127.84
att532	27,686	453,232	1,537.04	1,538.06	189,965	586.14	592.66	452,354	1,533.87	1,535.55	450,793	1,528.23	1,531.05	446,086	1,511.23	1,521.32
bayg29	1,610	2,946	82.98	83.45	1,730	7.43	6.21	2,807	74.33	80.71	2,598	61.35	75.59	2,382	47.98	43.73
bays29	2,020	3,698	83.08	83.47	2,184	8.10	9.55	3,490	72.75	79.26	3,293	63	67.08	3,008	48.92	48.27
berlin52	7,542	20,662	173.96	174.67	8,369	10.96	8.57	19,848	163.17	175.88	18,350	143.31	166.07	16,413	117.62	114.76
bier127	118,282	514,638	335.09	335.26	174,422	47.46	48.32	511,454	332.40	336.17	510,912	331.94	333.97	462,769	291.24	324.69
brazil58	25,395	82,863	226.30	227.78	30,504	20.12	19.76	78,775	210.20	224.25	72,396	185.08	216.07	65,549	158.12	184.31
brg180	1,950	655,445	33,512.58	33,495.64	36,228	1,757.83	1,709.74	652,221	33,347.22	33,424.62	649,134	33,188.91	33,409.23	634,928	32,460.38	32,693.85
burma14	3,323	3,588	7.97	8.19	3,323	-	-	3,607	8.54	8.56	3,480	4.72	4.57	3,392	2.07	1.78
ch130	6,110	36,988	505.36	506.15	10,994	79.93	80.80	37,286	510.25	509.63	36,012	489.39	504.86	34,861	470.56	498.71
ch150	6,528	43,754	570.24	571.68	12,326	88.81	86.13	43,697	569.37	572.92	43,340	563.91	566.31	41,071	529.14	554.66
d198	15,780	147,646	835.65	836.79	42,763	171	170.20	144,515	815.81	833.79	138,134	775.37	823.02	121,196	668.04	797.16
d493	35,002	399,081	1,040.17	1,040.86	176,502	404.26	399.21	398,290	1,037.91	1,038.20	396,978	1,034.16	1,035.53	395,606	1,030.24	1,032.31
d657	48,912	778,304	1,491.23	1,492.54	394,401	706.35	715.31	778,264	1,491.15	1,490.54	775,745	1,486	1,486.37	772,699	1,479.77	1,481.88
d1291	50,801	1,622,800	3,094.42	3,098.31	1,015,095	1,898.18	1,905.56	1,622,454	3,093.74	3,095.20	1,610,025	3,069.28	3,090.35	1,587,962	3,025.85	3,081.22
d1655	62,128	2,052,464	3,203.61	3,204.78	1,322,334	2,028.40	2,012.14	2,034,607	3,174.86	3,202.79	2,040,511	3,184.37	3,202.36	2,035,089	3,175.64	3,192.94
d2103	80,450	3,086,614	3,736.69	3,738.06	2,128,803	2,546.12	2,548.06	3,062,576	3,706.81	3,741.17	3,060,223	3,703.88	3,733.36	3,049,988	3,691.16	3,726.86
dantzig42	699	1,941	177.70	178.54	815	16.66	18.88	1,820	160.44	173.03	1,633	133.55	154.94	1,592	127.76	143.35
dsj1000	18,659,938	513,785,699	2,653.42	2,654.15	283,149,459	1,417.42	1,408.87	513,179,619	2,650.17	2,650.62	512,069,182	2,644.22	2,644.59	510,622,701	2,636.47	2,636.36
eil51	426	1,163	173.11	173.59	487	14.42	13.38	1,096	157.24	172.54	1,080	153.51	171.24	959	125.18	121.36
eil76	538	1,875	248.52	249.07	667	23.98	25.09	1,894	252	253.90	1,817	237.69	248.88	1,614	200	236.15
eil101	629	2,654	322.01	324.80	835	32.69	32.99	2,629	317.93	324.96	2,611	315.03	320.91	2,416	284.14	312.72
fl417	11,861	422,914	3,465.58	3,471.84	142,567	1,101.98	1,036.24	421,091	3,450.21	3,451.36	419,592	3,437.58	3,435.12	416,959	3,415.38	3,418.42
fl1400	20,127	1,549,452	7,598.38	7,601.35	757,886	3,665.52	3,652.05	1,541,459	7,558.66	7,596.42	1,545,871	7,580.58	7,595.22	1,534,428	7,523.73	7,541.47
fl1577	22,249	1,277,481	5,641.75	5,643.71	790,224	3,451.73	3,449.01	1,278,682	5,647.14	5,651.73	1,256,571	5,547.76	5,632.64	1,268,298	5,600.47	5,628.72
fl3795	28,772	3,448,467	11,885.50	11,897.30	2,720,572	9,355.63	9,349.97	3,448,093	11,884.20	11,889.65	3,447,784	11,883.12	11,882.56	3,439,429	11,854.09	11,861.09
fnl4461	182,566	8,046,941	4,307.69	4,310.10	6,592,306	3,510.92	3,510.73	8,037,697	4,302.63	4,309.09	8,028,435	4,297.55	4,306.41	8,027,551	4,297.07	4,298.81
fri26	937	1,594	70.16	71.02	984	5.03	2.99	1,511	61.21	67.13	1,396	48.99	52.93	1,303	39.01	33.94
gil262	2,378	22,854	861.07	860.62	7,931	233.52	227.65	22,831	860.11	860.74	22,670	853.34	858.92	22,451	844.10	852.27
gr17	2,085	2,501	19.95	20.77	2,091	0.30	0.24	2,433	16.67	16.12	2,281	9.41	8.56	2,261	8.45	6.62
gr21	2,707	4,228	56.20	57.68	2,838	4.84	-	4,110	51.84	50.68	3,772	39.36	42.15	3,467	28.09	26.97
gr24	1,272	2,072	62.88	63.40	1,334	4.83	4.44	1,965	54.50	62.42	1,820	43.11	43.47	1,720	35.25	31.21
gr48	5,046	14,136	180.14	181.99	5,654	12.05	12.24	13,676	171.03	182.18	12,487	147.47	173.35	11,676	131.40	144.22
gr96	55,209	278,553	404.54	405.74	77,878	41.06	41.10	277,104	401.92	406.19	269,196	387.59	396.92	239,054	333	389.48
gr120	6,942	39,706	471.97	472.13	10,909	57.15	56.70	38,866	459.87	473.82	38,926	460.73	467.63	36,120	420.31	461.68

Instance	BKS	Tuned-NLS (1-hour)			Random-NLS (1-hour)			Race-NLS (1-hour)			Race-NLS (2-hours)			Race-NLS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
gr137	69,853	491,106	603.06	605.66	129,152	84.89	86.93	485,224	594.64	605.58	478,973	585.69	600.95	434,506	522.03	592.49
gr202	40,160	228,338	468.57	469.06	79,687	98.42	101.12	227,593	466.71	468.28	223,620	456.82	463.66	216,580	439.29	460.70
gr229	134,602	1,144,107	749.99	750.60	375,162	178.72	180.89	1,143,450	749.50	752.36	1,136,161	744.09	744.65	1,103,966	720.17	737.80
gr431	171,414	2,129,384	1,142.25	1,142.91	822,621	379.90	383.95	2,126,933	1,140.82	1,141.56	2,118,743	1,136.04	1,137.70	2,107,331	1,129.38	1,130.92
gr666	294,358	4,635,078	1,474.64	1,473.80	2,302,329	682.15	683.17	4,581,755	1,456.52	1,470.51	4,573,826	1,453.83	1,464.05	4,554,997	1,447.43	1,456.89
hk48	11,461	32,301	181.83	182.87	12,918	12.71	10.60	31,026	170.71	181.68	29,170	154.52	175.54	25,343	121.13	132.75
kroA100	21,282	125,430	489.37	490.91	32,866	54.43	53.68	126,159	492.80	500.01	120,852	467.86	486.29	112,418	428.23	478.25
kroA150	26,524	201,714	660.50	661.94	54,473	105.37	104.62	202,145	662.12	663.10	200,896	657.41	656.45	185,270	598.50	650.28
kroA200	29,368	275,732	838.89	843.03	82,692	181.57	188.13	275,718	838.84	839.06	269,803	818.70	830.79	252,826	760.89	822.89
kroB100	22,141	124,454	462.10	464.86	33,444	51.05	51.72	124,156	460.75	466.24	120,696	445.13	462.31	113,508	412.66	444.01
kroB150	26,130	199,563	663.73	666.16	55,978	114.23	117.31	197,007	653.95	668.28	196,348	651.43	660.52	190,657	629.65	653.66
kroB200	29,437	271,252	821.47	822.21	78,617	167.07	162.91	270,589	819.21	820.31	268,476	812.03	813.81	257,046	773.21	805.28
kroC100	20,749	124,434	499.71	501.44	31,638	52.48	48.63	124,111	498.15	502.50	121,806	487.05	498.66	115,186	455.14	490.51
kroD100	21,294	121,528	470.72	471.65	33,344	56.59	55.55	120,528	466.02	469.49	114,806	439.15	465.95	107,887	406.65	459.71
kroE100	22,068	126,697	474.12	475.31	33,747	52.92	49.65	126,484	473.15	479.53	117,882	434.18	472.34	111,245	404.10	466.61
lin105	14,379	91,112	533.65	534.62	23,704	64.85	65.59	90,681	530.65	539.76	88,091	512.64	528.64	80,701	461.24	520.79
lin318	42,029	509,617	1,112.54	1,113.59	187,898	347.07	351.57	509,908	1,113.23	1,113.52	507,497	1,107.49	1,106.17	504,699	1,100.84	1,103.02
nrw1379	56,638	1,327,930	2,244.59	2,247.60	778,425	1,274.39	1,267.84	1,329,614	2,247.57	2,250.28	1,323,201	2,236.24	2,239.85	1,323,730	2,237.18	2,239.38
p654	34,643	1,792,343	5,073.75	5,076.53	688,458	1,887.29	1,868.68	1,781,609	5,042.77	5,051.16	1,780,848	5,040.57	5,041.80	1,772,487	5,016.44	5,018.03
pa561	2,763	33,368	1,107.66	1,108.85	17,035	516.54	516.54	33,342	1,106.73	1,106.95	33,206	1,101.81	1,102.59	32,908	1,091.01	1,098.59
pcb442	50,778	687,863	1,254.65	1,254.91	284,872	461.01	457.06	686,654	1,252.27	1,253.06	684,796	1,248.61	1,249.80	682,232	1,243.56	1,244.68
pcb1173	56,892	1,311,540	2,205.31	2,207.08	739,852	1,200.45	1,202.10	1,315,045	2,211.48	2,211.87	1,292,419	2,171.71	2,203.76	1,291,302	2,169.74	2,195.87
pcb3038	137,694	5,187,955	3,667.74	3,668.07	3,966,188	2,780.44	2,768.87	5,049,748	3,567.37	3,619.87	5,084,620	3,592.70	3,629.87	5,121,541	3,619.51	3,646.53
pr76	108,159	426,179	294.03	294.51	138,836	28.36	28.41	418,146	286.60	300.54	395,429	265.60	289.26	368,894	241.07	273.79
pr107	44,303	406,015	816.45	818.72	94,470	113.24	120.56	402,070	807.55	819.02	384,230	767.28	807.35	336,429	659.38	776.49
pr124	59,030	537,642	810.79	812.69	121,012	105	100.88	530,779	799.17	812.85	509,658	763.39	799.88	484,761	721.21	791.91
pr136	96,772	647,730	569.34	571.79	175,882	81.75	80.04	652,199	573.95	574.38	641,400	562.79	566.97	610,452	530.81	558.21
pr144	58,537	645,600	1,002.89	1,003.87	154,377	163.73	165.75	643,091	998.61	998.32	640,163	993.60	996.77	591,772	910.94	985.47
pr152	73,682	832,718	1,030.15	1,033.10	204,969	178.18	175.16	825,355	1,020.16	1,037.45	827,789	1,023.46	1,022.43	774,601	951.28	1,010.49
pr226	80,369	1,404,761	1,647.89	1,650.34	394,029	390.27	398.01	1,405,030	1,648.22	1,649.57	1,396,205	1,637.24	1,638.05	1,362,480	1,595.28	1,618.67
pr264	49,135	906,097	1,744.10	1,745.93	242,958	394.47	405.22	902,794	1,737.37	1,742.34	894,769	1,721.04	1,725.72	879,927	1,690.84	1,713.05
pr299	48,191	637,374	1,222.60	1,223.59	220,371	357.29	365	634,482	1,216.60	1,218	632,909	1,213.33	1,215.55	618,333	1,183.09	1,207.06
pr439	107,217	1,679,528	1,466.48	1,469	691,528	544.98	538.81	1,676,748	1,463.88	1,463	1,670,337	1,457.90	1,458.79	1,649,616	1,438.58	1,450.01
pr1002	259,045	5,955,109	2,198.87	2,200.47	3,072,899	1,086.24	1,089.22	5,705,703	2,102.59	2,139.75	5,580,572	2,054.29	2,057.85	5,374,216	1,974.63	1,949.97
pr2392	378,032	14,541,962	3,746.75	3,749.04	10,340,804	2,635.43	2,634.74	14,546,854	3,748.05	3,748.23	14,517,090	3,740.17	3,745.73	14,448,695	3,722.08	3,737.60
rat99	1,211	6,163	408.92	410.94	1,774	46.47	45.38	6,122	405.52	417.26	6,041	398.81	409.08	5,428	348.22	402.23
rat195	2,323	18,403	692.21	693.65	5,804	149.87	155.62	18,097	679.05	692.42	18,072	677.95	688.38	17,658	660.12	684.72
rat575	6,773	101,483	1,398.34	1,399.20	46,786	590.78	591.07	101,268	1,395.17	1,395.14	100,518	1,384.09	1,388.09	99,369	1,367.13	1,383.73
rat783	8,806	162,805	1,748.80	1,751.16	84,999	865.24	860.54	162,702	1,747.63	1,753.34	162,177	1,741.67	1,744.48	160,860	1,726.71	1,733.43
rd100	7,910	42,460	436.79	437.83	11,516	45.59	45.56	42,597	438.53	440.42	40,854	416.48	434.94	36,908	366.59	426.06
rd400	15,281	187,130	1,124.59	1,124.33	76,999	403.89	406.41	186,806	1,122.47	1,125.74	186,119	1,117.97	1,118.71	185,552	1,114.26	1,115.87
rl1304	252,948	8,761,692	3,363.83	3,367.41	5,129,882	1,928.04	1,938.01	8,783,147	3,372.31	3,368.80	8,760,831	3,363.49	3,368.31	8,747,203	3,358.10	3,358.49
rl1323	270,199	9,167,123	3,292.73	3,294.64	5,385,992	1,893.34	1,858.07	9,166,716	3,292.58	3,293.74	9,141,237	3,283.15	3,282.87	9,132,332	3,279.85	3,282.69
rl1889	316,536	13,978,274	4,316.01	4,316.02	9,254,733	2,823.75	2,821.21	13,997,105	4,321.96	4,323.13	13,977,658	4,315.82	4,314.18	13,942,069	4,304.58	4,302.50

Instance	BKS	Tuned-NLS (1-hour)			Random-NLS (1-hour)			Race-NLS (1-hour)			Race-NLS (2-hours)			Race-NLS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
rl5915	565,530	41,358,111	7,213.16	7,212.64	35,772,802	6,225.54	6,230.20	41,360,127	7,213.52	7,213.26	41,326,998	7,207.66	7,209.81	41,289,120	7,200.96	7,203.64
rl5934	556,045	41,022,514	7,277.55	7,280.32	35,293,340	6,247.21	6,239.97	41,001,512	7,273.78	7,274.17	40,980,289	7,269.96	7,268.75	40,903,999	7,256.24	7,258.42
si175	21,407	43,831	104.75	104.64	26,291	22.82	23.23	43,730	104.28	104.31	43,634	103.83	103.89	42,493	98.50	103.24
si535	48,450	146,811	203.02	203.01	93,202	92.37	93.89	146,604	202.59	202.63	146,230	201.82	202.04	145,896	201.13	201.57
si1032	92,650	347,391	274.95	275.01	238,693	157.63	157.98	347,183	274.73	275.13	346,762	274.27	274.57	346,289	273.76	273.89
st70	675	2,649	292.45	293.93	872	29.21	32	2,590	283.67	294	2,397	255.09	283.70	2,227	229.93	276.89
swiss42	1,273	3,213	152.43	153.14	1,448	13.76	14.49	3,087	142.54	150.75	2,900	127.81	140.77	2,623	106.07	121.05
ts225	126,643	1,348,807	965.05	967.29	440,729	248.01	252.52	1,336,394	955.25	961.47	1,329,987	950.19	959.78	1,319,590	941.98	949.76
tsp225	3,916	34,740	787.13	787.28	10,801	175.82	171.50	34,698	786.06	787.65	34,310	776.16	780.94	32,682	734.59	776.10
u159	42,080	357,920	750.57	751.88	94,469	124.50	128.82	352,785	738.37	749.57	349,800	731.27	742.59	347,238	725.19	736.81
u574	36,905	609,957	1,552.78	1,553.29	288,151	680.79	669.52	607,215	1,545.35	1,549.13	598,403	1,521.47	1,531.75	597,936	1,520.20	1,536.92
u724	41,910	793,310	1,792.89	1,793.40	398,370	850.54	850.37	791,253	1,787.98	1,791.70	787,410	1,778.81	1,780.76	785,501	1,774.26	1,776.28
u1060	224,094	6,191,782	2,663.03	2,664.29	3,124,758	1,294.40	1,291.84	6,179,035	2,657.34	2,665.99	6,170,029	2,653.32	2,655.95	6,150,660	2,644.68	2,649.10
u1432	152,970	3,699,425	2,318.40	2,320	2,216,553	1,349.01	1,345.99	3,699,798	2,318.64	2,318.91	3,682,136	2,307.10	2,315.11	3,622,439	2,268.07	2,301.94
u1817	57,201	1,998,303	3,393.48	3,396.60	1,296,324	2,166.26	2,159.20	1,996,976	3,391.16	3,393.21	1,987,554	3,374.68	3,389.53	1,984,152	3,368.74	3,379.55
u2152	64,253	2,399,852	3,635	3,636.45	1,629,499	2,436.07	2,428.80	2,397,608	3,631.51	3,640.38	2,386,129	3,613.65	3,629.91	2,376,622	3,598.85	3,620.58
u2319	234,256	5,723,866	2,343.42	2,344.22	4,076,927	1,640.37	1,637.48	5,720,634	2,342.04	2,346.04	5,715,444	2,339.83	2,342.61	5,657,667	2,315.16	2,334.73
ulysses16	6,859	7,560	10.22	10.18	6,873	0.21	0.16	7,509	9.47	9.43	7,335	6.94	7.14	7,183	4.73	3.91
ulysses22	7,013	9,595	36.81	37.10	7,083	0.99	1.06	9,079	29.46	32.32	8,744	24.68	25.35	8,259	17.77	16.75
vm1084	239,297	7,910,262	3,205.63	3,208.52	4,086,381	1,607.66	1,604.92	7,909,357	3,205.25	3,207.64	7,907,204	3,204.35	3,208.48	7,885,739	3,195.38	3,197.27
vm1748	336,556	14,112,261	4,093.14	4,095.10	9,059,960	2,591.96	2,585.56	14,095,073	4,088.03	4,091.79	14,100,991	4,089.79	4,090.86	14,075,190	4,082.12	4,082.72

Table C.2: The table presents the complete results for the TSP with Local Search. Each row represents a problem instance, and “BKS” stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature.

Instance	BKS	Tuned-LS (1-hour)			Random-LS (1-hour)			Race-LS (1-hour)			Race-LS (2-hours)			Race-LS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
a280	2,579	2,823	9.46	5.35	2,709	5.03	5.02	2,917	13.11	13.03	2,890	12.06	11.83	2,895	12.25	11.83
ali535	202,339	239,809	18.52	10.67	221,158	9.30	9.25	1,051,897	419.87	398.40	989,723	389.14	390.35	981,871	385.26	383
att48	10,628	10,628	-	-	10,628	-	-	10,664	0.34	0.33	10,654	0.25	0.29	10,643	0.14	0.09
att532	27,686	33,381	20.57	8.73	29,630	7.02	6.76	132,692	379.28	398.11	138,191	399.14	390.09	128,378	363.69	366.61
bayg29	1,610	1,610	-	-	1,610	-	-	1,610	-	-	1,610	-	-	1,610	-	-
bays29	2,020	2,020	-	-	2,020	-	-	2,020	-	-	2,020	-	-	2,020	-	-
berlin52	7,542	7,542	-	-	7,542	-	-	7,637	1.26	1.15	7,598	0.74	0.72	7,572	0.39	0.37
bier127	118,282	119,578	1.10	1.11	119,524	1.05	1.04	123,865	4.72	4.73	123,463	4.38	4.36	122,724	3.76	3.73
brazil58	25,395	25,395	-	-	25,395	-	-	25,595	0.79	0.84	25,511	0.46	0.43	25,477	0.32	0.27
brg180	1,950	2,034	4.29	4.10	2,042	4.74	5.13	2,063	5.78	5.64	2,045	4.85	5.13	2,026	3.89	4.10
burma14	3,323	3,323	-	-	3,323	-	-	3,323	-	-	3,323	-	-	3,323	-	-
ch130	6,110	6,191	1.33	1.30	6,196	1.40	1.42	6,461	5.74	5.84	6,433	5.29	5.36	6,411	4.92	5.03
ch150	6,528	6,658	1.99	2.01	6,659	2.01	2.11	7,057	8.11	8.35	6,994	7.14	7.34	6,971	6.78	6.88
d198	15,780	16,073	1.86	1.83	16,077	1.88	1.82	16,567	4.99	5.08	16,543	4.83	4.86	16,514	4.65	4.67
d493	35,002	37,780	7.94	6.88	37,144	6.12	6.07	109,611	213.16	213.53	104,334	198.08	196.27	104,633	198.93	194.87
d657	48,912	129,473	164.71	177.84	52,787	7.92	7.88	286,250	485.23	495.40	276,984	466.29	460.85	268,225	448.38	452.02
d1291	50,801	777,139	1,429.77	1,467.50	56,326	10.88	11.18	863,911	1,600.58	1,562.21	851,666	1,576.47	1,570.87	825,890	1,525.74	1,531.16
d1655	62,128	1,071,977	1,625.43	1,660.44	1,066,091	1,615.96	1,581.88	1,056,322	1,600.23	1,587.41	1,045,625	1,583.02	1,575.38	996,580	1,504.08	1,517.94
d2103	80,450	1,878,122	2,234.52	2,215.44	1,732,140	2,053.06	2,121.15	1,632,269	1,928.92	1,886.42	1,594,031	1,881.39	1,862.33	1,505,913	1,771.86	1,755.20
dantzig42	699	699	-	-	699	-	-	701	0.30	0.14	700	0.20	0.14	699	0.02	-
dsj1000	18,659,938	170,564,041	814.07	767.67	20,176,213	8.13	7.98	211,799,529	1,035.05	1,037.54	204,727,062	997.15	1,005.86	186,286,949	898.33	882.98
eil51	426	426	-	-	426	0.02	-	431	1.06	0.94	430	0.88	0.94	428	0.58	0.59
eil76	538	543	0.90	0.93	543	0.88	0.93	558	3.62	3.81	555	3.16	3.16	554	2.88	2.97
eil101	629	641	1.96	2.07	642	2.05	2.15	660	4.91	5.25	658	4.54	4.61	655	4.18	4.29
fl417	11,861	12,376	4.34	4.20	12,318	3.86	3.79	46,763	294.26	247.92	43,229	264.46	240.50	39,259	230.99	206.55
fl1400	20,127	611,736	2,939.38	2,993.65	617,126	2,966.16	3,480.93	533,648	2,551.40	2,524.84	507,191	2,419.95	2,488.23	404,566	1,910.07	1,918.41
fl1577	22,249	690,733	3,004.56	2,979.08	658,710	2,860.63	2,916.57	725,141	3,159.21	3,161.95	724,632	3,156.92	3,121.96	691,338	3,007.28	3,010.40
fl3795	28,772	2,293,296	7,870.58	7,941.07	2,172,836	7,451.91	7,253.39	2,042,038	6,997.31	6,920.85	1,968,084	6,740.28	6,801.38	1,914,632	6,554.50	6,614.61
fnl4461	182,566	5,223,194	2,760.99	2,593.76	5,425,326	2,871.71	2,784.69	4,453,553	2,339.42	2,333.39	4,371,986	2,294.74	2,284.37	4,242,786	2,223.97	2,221.11
fri26	937	937	-	-	937	-	-	937	-	-	937	-	-	937	-	-
gil262	2,378	2,498	5.03	4.98	2,493	4.85	4.88	2,668	12.20	12.24	2,649	11.38	11.52	2,637	10.88	10.89
gr17	2,085	2,085	-	-	2,085	-	-	2,085	-	-	2,085	-	-	2,085	-	-
gr21	2,707	2,707	-	-	2,707	-	-	2,707	-	-	2,707	-	-	2,707	-	-
gr24	1,272	1,272	-	-	1,272	-	-	1,272	-	-	1,272	-	-	1,272	-	-
gr48	5,046	5,046	-	-	5,046	-	-	5,069	0.45	0.40	5,066	0.41	0.40	5,059	0.26	0.24
gr96	55,209	55,483	0.50	0.54	55,505	0.54	0.55	57,203	3.61	3.67	56,985	3.22	3.34	56,829	2.93	2.99
gr120	6,942	7,051	1.58	1.54	7,043	1.45	1.53	7,381	6.33	6.50	7,364	6.08	5.88	7,310	5.29	5.40

Instance	BKS	Tuned-LS (1-hour)			Random-LS (1-hour)			Race-LS (1-hour)			Race-LS (2-hours)			Race-LS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
gr137	69,853	70,811	1.37	1.35	70,798	1.35	1.49	74,581	6.77	7.13	74,292	6.36	6.40	73,763	5.60	5.72
gr202	40,160	41,467	3.25	3.25	41,465	3.25	3.28	43,220	7.62	7.84	43,019	7.12	7.10	42,902	6.83	6.89
gr229	134,602	139,084	3.33	3.41	138,956	3.23	3.30	145,866	8.37	8.37	145,641	8.20	8.19	144,932	7.67	7.78
gr431	171,414	182,478	6.45	6.20	181,901	6.12	5.86	424,769	147.80	151.87	385,253	124.75	122.03	386,702	125.60	127.80
gr666	294,358	884,628	200.53	178.62	322,861	9.68	9.70	1,751,884	495.15	495.59	1,694,825	475.77	471.41	1,720,468	484.48	492.43
hk48	11,461	11,461	-	-	11,461	-	-	11,565	0.90	0.93	11,555	0.82	0.68	11,518	0.50	0.41
kroA100	21,282	21,301	0.09	0.05	21,300	0.08	0.07	22,243	4.52	4.61	22,196	4.29	4.32	22,115	3.91	3.89
kroA150	26,524	26,979	1.72	1.70	26,973	1.69	1.74	28,302	6.70	6.92	28,262	6.55	6.73	28,147	6.12	6.29
kroA200	29,368	30,112	2.53	2.46	30,076	2.41	2.28	31,689	7.90	7.97	31,666	7.82	8.01	31,528	7.35	7.32
kroB100	22,141	22,271	0.59	0.63	22,279	0.62	0.63	23,134	4.49	4.57	23,021	3.97	4.05	22,961	3.70	3.72
kroB150	26,130	26,558	1.64	1.68	26,513	1.47	1.44	27,858	6.61	6.50	27,774	6.29	6.35	27,632	5.75	5.71
kroB200	29,437	30,409	3.30	3.34	30,432	3.38	3.47	32,031	8.81	8.90	31,972	8.61	8.69	31,778	7.95	8.02
kroC100	20,749	20,777	0.13	0.10	20,768	0.09	0.10	21,898	5.54	5.71	21,767	4.90	5.05	21,654	4.36	4.49
kroD100	21,294	21,481	0.88	0.87	21,499	0.96	0.92	22,289	4.67	4.72	22,197	4.24	4.21	22,149	4.01	4.02
kroE100	22,068	22,224	0.71	0.73	22,226	0.71	0.73	23,063	4.51	4.61	22,984	4.15	4.10	22,925	3.88	3.99
lin105	14,379	14,431	0.36	0.35	14,438	0.41	0.46	15,096	4.98	4.96	15,031	4.54	4.61	14,980	4.18	4.21
lin318	42,029	44,206	5.18	4.98	44,012	4.72	4.67	51,730	23.08	13.76	51,020	21.39	13.16	48,239	14.78	13.33
nrw1379	56,638	622,510	999.10	913.63	529,149	834.26	865.22	649,876	1,047.42	1,035.49	640,778	1,031.36	1,030.89	624,557	1,002.72	1,009.40
p654	34,643	102,416	195.63	148.32	36,554	5.52	5.14	384,775	1,010.69	908.56	266,058	668	647.86	224,531	548.13	531.52
pa561	2,763	3,122	12.99	9.99	3,008	8.87	8.98	10,806	291.08	294.95	11,115	302.27	296.45	10,889	294.10	287.40
pcb442	50,778	54,500	7.33	7.43	54,316	6.97	7.21	140,874	177.43	182.08	142,212	180.07	177.54	134,193	164.27	153.66
pcb1173	56,892	564,665	892.52	899.88	280,732	393.45	434.05	642,551	1,029.42	1,014.20	631,975	1,010.83	1,000.50	611,236	974.38	977.64
pcb3038	137,694	3,255,029	2,263.96	2,144.67	3,156,470	2,192.38	2,191.48	2,862,243	1,978.70	1,961.33	2,777,022	1,916.81	1,903.90	2,662,879	1,833.91	1,830.47
pr76	108,159	108,268	0.10	0.11	108,258	0.09	0.11	110,507	2.17	2.26	110,251	1.93	2.11	110,014	1.72	1.73
pr107	44,303	44,682	0.85	0.85	44,633	0.74	0.76	45,993	3.81	3.80	45,935	3.68	3.72	45,696	3.14	3.20
pr124	59,030	59,172	0.24	0.28	59,146	0.20	0.22	62,661	6.15	6.17	62,080	5.17	5.50	61,972	4.98	5.09
pr136	96,772	98,169	1.44	1.38	98,287	1.57	1.57	101,703	5.10	5.17	101,151	4.53	4.56	100,656	4.01	3.93
pr144	58,537	58,617	0.14	0.14	58,611	0.13	0.14	61,433	4.95	5	61,143	4.45	4.63	60,925	4.08	3.99
pr152	73,682	74,269	0.80	0.81	74,233	0.75	0.79	77,993	5.85	5.98	77,743	5.51	5.76	77,520	5.21	5.33
pr226	80,369	81,006	0.79	0.75	80,889	0.65	0.61	87,940	9.42	9.58	87,584	8.98	8.77	86,654	7.82	8.03
pr264	49,135	50,369	2.51	2.21	50,427	2.63	2.70	55,017	11.97	12.13	54,316	10.54	10.80	54,085	10.07	10.19
pr299	48,191	50,425	4.63	4.55	50,370	4.52	4.50	55,455	15.07	14.09	55,061	14.26	13.86	54,768	13.65	13.77
pr439	107,217	114,646	6.93	7.04	113,488	5.85	5.75	387,191	261.13	276.65	371,909	246.88	246.07	362,914	238.49	244.59
pr1002	259,045	1,970,797	660.79	654.82	1,117,841	331.52	406.40	2,566,505	890.76	869.68	2,531,471	877.23	882.08	2,461,996	850.41	854.60
pr2392	378,032	8,768,735	2,219.57	2,094.84	9,069,370	2,299.10	2,275.88	7,828,307	1,970.81	1,913.54	7,535,051	1,893.23	1,886.42	7,339,025	1,841.38	1,837.68
rat99	1,211	1,223	0.97	0.99	1,224	1.05	1.07	1,283	5.95	5.90	1,273	5.09	5.04	1,268	4.72	4.87
rat195	2,323	2,415	3.96	3.85	2,416	4	4.09	2,553	9.92	9.99	2,533	9.02	8.91	2,519	8.42	8.33
rat575	6,773	10,100	49.12	43.21	7,302	7.80	7.72	32,838	384.83	373.70	32,665	382.29	387.66	31,671	367.61	365.54
rat783	8,806	45,662	418.53	438.92	9,531	8.23	8.18	66,427	654.33	660.22	65,953	648.96	647.94	64,468	632.09	636.32
rd100	7,910	7,953	0.54	0.56	7,954	0.56	0.56	8,313	5.09	5.08	8,276	4.63	4.60	8,243	4.22	4.21
rd400	15,281	18,402	20.42	6.11	16,191	5.96	5.94	31,317	104.94	96.41	32,341	111.64	110.32	32,029	109.60	108.29
rl1304	252,948	3,968,655	1,468.96	1,441.64	2,786,189	1,001.49	1,391.10	4,388,993	1,635.14	1,607.59	4,349,325	1,619.45	1,632.09	4,240,418	1,576.40	1,576.27
rl1323	270,199	3,849,719	1,324.77	1,361.48	2,657,049	883.37	1,141.45	4,487,714	1,560.89	1,581.85	4,355,100	1,511.81	1,492.33	4,336,656	1,504.99	1,504.08
rl1889	316,536	7,944,088	2,409.69	2,386.05	7,561,388	2,288.79	2,156.26	7,057,043	2,129.46	2,078.06	6,944,211	2,093.81	2,083.71	6,625,524	1,993.13	1,981.08

PUC-Rio - Certificação Digital Nº 2021588/CA

Instance	BKS	Tuned-LS (1-hour)			Random-LS (1-hour)			Race-LS (1-hour)			Race-LS (2-hours)			Race-LS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
rl5915	565,530	29,153,558	5,055.09	5,025.61	27,891,064	4,831.85	4,600.91	24,252,431	4,188.44	4,136.87	23,476,902	4,051.31	4,002.74	23,130,931	3,990.13	3,955.49
rl5934	556,045	28,847,062	5,087.90	4,994.40	27,333,253	4,815.65	4,697.30	24,695,030	4,341.19	4,273.66	23,839,788	4,187.38	4,132.22	23,119,525	4,057.85	4,020.98
si175	21,407	21,478	0.33	0.33	21,483	0.36	0.37	21,706	1.40	1.41	21,675	1.25	1.27	21,663	1.20	1.19
si535	48,450	49,009	1.15	0.93	48,875	0.88	0.89	70,930	46.40	47.83	68,224	40.81	41.59	68,375	41.12	40.82
si1032	92,650	136,199	47	41.88	93,245	0.64	0.67	219,671	137.10	137.40	203,438	119.58	124.63	196,173	111.74	117.44
st70	675	675	0.04	-	675	0.05	-	690	2.19	2.22	689	2.02	2	686	1.65	1.78
swiss42	1,273	1,273	-	-	1,273	-	-	1,274	0.11	-	1,273	0.03	-	1,273	-	-
ts225	126,643	128,366	1.36	1.26	128,316	1.32	1.27	136,064	7.44	7.65	134,947	6.56	6.54	134,118	5.90	5.72
tsp225	3,916	4,101	4.71	4.70	4,102	4.75	4.75	4,307	9.99	10.24	4,579	16.94	9.67	4,261	8.80	8.86
u159	42,080	42,819	1.76	1.88	42,857	1.85	1.93	45,873	9.01	9.31	45,613	8.39	8.61	45,371	7.82	7.90
u574	36,905	63,173	71.18	55.51	39,764	7.75	7.76	197,936	436.34	433.32	196,598	432.71	439.89	185,886	403.69	409.07
u724	41,910	166,259	296.70	300.35	45,261	7.99	7.85	297,072	608.83	614.69	291,977	596.68	599.67	280,604	569.54	570.51
u1060	224,094	2,342,540	945.34	877.52	1,313,061	485.94	493.61	2,789,621	1,144.84	1,136.99	2,745,505	1,125.16	1,117.42	2,719,532	1,113.57	1,117
u1432	152,970	1,851,414	1,110.31	1,079.03	1,595,156	942.79	938.32	1,818,762	1,088.97	1,082.60	1,782,764	1,065.43	1,049.76	1,718,099	1,023.16	1,014.33
u1817	57,201	1,020,588	1,684.21	1,650.69	1,026,992	1,695.41	1,636.10	1,038,087	1,714.80	1,697.37	1,010,365	1,666.34	1,648.46	984,020	1,620.28	1,611.54
u2152	64,253	1,311,752	1,941.54	1,787.57	1,410,858	2,095.78	2,005.47	1,257,503	1,857.11	1,837.79	1,223,013	1,803.43	1,781.69	1,211,226	1,785.09	1,776.41
u2319	234,256	3,469,780	1,381.19	1,351.47	3,385,743	1,345.32	1,305.24	3,082,514	1,215.87	1,208.67	3,085,591	1,217.19	1,204.90	2,991,410	1,176.98	1,177.08
ulysses16	6,859	6,859	-	-	6,859	-	-	6,859	-	-	6,859	-	-	6,859	-	-
ulysses22	7,013	7,013	-	-	7,013	-	-	7,013	-	-	7,013	-	-	7,013	-	-
vm1084	239,297	3,034,359	1,168.03	1,137.95	1,721,587	619.44	357.54	3,492,535	1,359.50	1,336.92	3,287,450	1,273.80	1,264.27	3,250,791	1,258.48	1,246.35
vm1748	336,556	7,683,557	2,183	2,233.61	7,305,468	2,070.65	1,934.22	7,046,649	1,993.75	1,973.05	6,799,316	1,920.26	1,916.10	6,573,849	1,853.27	1,843.32

D
Complete Results for the Set Covering Problem

Table D.1: The table presents the complete results for the SCP without Local Search. Each row represents a problem instance, and "BKS" stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature.

Instance	BKS	Tuned-NLS (1-hour)			Random-NLS (1-hour)			Race-NLS (1-hour)			Race-NLS (2-hours)			Race-NLS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
scp41	429	569	32.69	31.47	1,600	273.01	258.28	12,602	2,837.57	2,937.53	9,626	2,143.89	2,385.43	7,292	1,599.85	1,517.25
scp42	512	703	37.28	36.43	1,820	255.52	205.37	14,071	2,648.26	2,696.09	9,359	1,727.86	1,877.34	7,714	1,406.64	1,406.25
scp43	516	700	35.56	35.66	1,743	237.86	215.41	14,111	2,634.65	2,706.30	9,833	1,805.59	1,958.04	6,481	1,156.03	928.29
scp44	494	651	31.69	29.35	1,645	233.06	226.01	12,974	2,526.25	2,617.31	9,428	1,808.56	1,850.91	7,864	1,491.81	1,733.70
scp45	512	685	33.78	30.76	1,619	216.18	217.09	13,851	2,605.19	2,583.20	9,224	1,701.58	1,870.61	7,327	1,331.07	1,246.48
scp46	560	711	26.90	26.61	1,922	243.17	216.88	13,724	2,350.74	2,292.32	10,476	1,770.76	2,166.96	7,354	1,213.15	1,162.23
scp47	430	605	40.76	42.44	1,780	314.05	253.14	12,585	2,826.75	2,938.02	8,789	1,944.04	2,208.14	6,961	1,518.78	1,403.49
scp48	492	647	31.50	30.18	1,790	263.88	250.91	13,850	2,714.95	2,733.03	10,933	2,122.15	2,584.76	7,503	1,425.01	1,433.94
scp49	641	843	31.50	30.50	2,431	279.31	174.73	14,444	2,153.38	2,312.64	10,013	1,462.11	1,465.68	7,035	997.48	933.54
scp410	514	661	28.52	28.11	1,821	254.27	252.63	12,362	2,304.99	2,392.90	10,164	1,877.46	2,100.19	7,215	1,303.73	1,250.78
scp51	253	2,296	807.42	837.15	12,003	4,644.32	4,413.24	31,902	12,509.68	12,879.25	26,108	10,219.43	11,434.78	20,301	7,924.07	8,607.71
scp52	302	2,256	646.96	646.69	12,404	4,007.35	3,957.62	33,207	10,895.58	11,468.87	27,258	8,925.81	10,277.48	21,623	7,059.86	7,243.87
scp53	226	2,315	924.19	889.38	12,015	5,216.17	5,074.34	30,331	13,320.91	13,877.21	26,607	11,672.84	12,942.48	20,314	8,888.56	9,808.41
scp54	242	2,181	801.12	753.93	12,567	5,093.10	4,411.78	32,692	13,409.30	13,675.62	26,388	10,804.24	12,294.42	21,755	8,889.87	9,359.50
scp55	211	2,289	984.71	946.68	12,061	5,616.05	5,606.40	34,132	16,076.45	16,513.74	26,541	12,478.72	13,595.97	21,006	9,855.49	10,262.09
scp56	213	2,260	960.86	952.82	12,509	5,772.63	5,486.15	32,465	15,141.58	15,600.47	27,504	12,812.88	14,382.16	20,092	9,332.65	10,316.67
scp57	293	2,284	679.59	681.06	12,191	4,060.69	3,981.06	32,573	11,016.99	11,321.84	27,933	9,433.38	10,481.06	19,759	6,643.57	6,912.63
scp58	288	2,156	648.72	642.36	12,397	4,204.40	4,140.80	32,414	11,154.99	11,725.87	26,464	9,088.74	9,975.17	18,269	6,243.45	6,677.95
scp59	279	2,237	701.84	663.08	11,852	4,147.87	4,286.38	33,728	11,988.92	12,064.34	26,338	9,340.17	10,431.90	20,689	7,315.54	8,013.62
scp61	138	306	122.08	122.10	2,480	1,697.29	624.28	12,931	9,270.32	9,521.01	9,590	6,849.01	7,022.46	6,934	4,924.98	4,887.68
scp62	146	318	117.76	108.90	1,194	717.79	546.92	12,850	8,701.08	8,773.63	10,036	6,774.23	8,125.00	7,120	4,776.52	4,646.58
scp63	145	292	101.72	82.76	1,152	694.83	725.17	13,166	8,980.06	8,982.07	9,428	6,402.18	6,995.86	7,195	4,862.07	4,848.97
scp64	131	262	99.90	81.30	976	644.71	670.61	13,015	9,835.21	9,750.38	9,943	7,490.11	7,951.15	6,854	5,132.12	5,806.87
scp65	161	306	90.27	90.99	1,207	649.83	555.28	12,719	7,799.70	7,822.98	9,847	6,016.41	6,187.58	6,793	4,119.02	3,850.93
scp510	265	2,287	763.04	764.34	13,065	4,830.28	4,342.26	34,873	13,059.65	13,620.75	29,354	10,976.87	12,363.21	22,062	8,225.22	8,696.42
scpa1	253	4,348	1,618.55	1,620.16	49,494	19,462.82	19,527.67	53,708	21,128.44	22,109.29	46,140	18,137.12	19,264.03	37,892	14,876.92	14,262.65
scpa2	252	4,250	1,586.71	1,578.57	49,538	19,557.88	19,615.48	53,096	20,969.94	21,435.52	43,528	17,173.05	18,448.81	38,923	15,345.55	15,576.39
scpa3	232	4,111	1,671.95	1,659.27	48,503	20,806.45	20,675.65	51,666	22,169.70	23,705.60	42,375	18,165.01	20,874.14	36,903	15,806.50	16,874.14
scpa4	234	4,241	1,712.42	1,704.27	49,476	21,043.79	20,893.59	53,940	22,951.25	24,352.99	47,478	20,189.80	22,096.37	37,232	15,811.00	16,656.41
scpa5	236	3,954	1,575.56	1,550.42	48,506	20,453.45	20,275.64	52,495	22,143.51	22,545.55	47,367	19,970.75	21,099.58	39,299	16,552.01	17,541.10
scpb1	69	4,229	6,029.47	5,996.38	49,746	71,995.31	71,371.01	52,418	75,868.67	80,949.28	47,550	68,812.65	75,739.86	41,442	59,961.51	68,549.28
scpb2	76	4,218	5,449.34	5,422.37	50,105	65,827.28	65,979.61	54,122	71,112.66	72,090.79	47,458	62,344.31	67,040.13	40,468	53,147.76	55,114.47
scpb3	80	4,177	5,120.67	5,078.12	49,686	62,007.38	61,398.12	53,687	67,009.07	68,810.00	47,271	58,988.17	65,430.62	40,025	49,931.75	53,663.75
scpb4	79	4,049	5,025.70	4,956.96	48,215	60,931.48	61,352.53	52,542	66,408.26	69,391.14	46,024	58,158.78	62,922.78	39,487	49,883.28	51,248.10
scpb5	72	4,144	5,655.60	5,594.44	49,052	68,028.15	67,727.78	51,656	71,644.44	76,027.08	47,034	65,225.52	70,854.86	38,750	53,718.98	52,107.64
scpc1	227	8,263	3,540.04	3,443.83	82,698	36,330.63	36,229.30	77,821	34,182.28	35,472.25	69,334	30,443.61	31,994.27	62,529	27,445.80	28,040.97
scpc2	219	8,290	3,685.22	3,692.92	80,233	36,536.23	36,625.57	75,088	34,186.68	35,312.79	64,803	29,490.29	31,355.25	61,360	27,918.32	30,025.11

Instance	BKS	Tuned-NLS (1-hour)			Random-NLS (1-hour)			Race-NLS (1-hour)			Race-NLS (2-hours)			Race-NLS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
scpc3	243	8,338	3,331.21	3,379.42	82,902	34,016.15	34,058.85	78,684	32,280.25	33,318.52	69,442	28,477.06	30,328.60	62,005	25,416.48	27,852.67
scpc4	219	8,059	3,579.83	3,589.50	81,225	36,988.95	37,084.93	76,530	34,845.29	35,939.50	66,733	30,371.80	33,051.14	63,952	29,101.92	31,513.70
scpc5	215	8,271	3,746.87	3,810.23	82,126	38,098.28	38,050.93	75,218	34,885.27	37,444.65	68,249	31,643.55	33,986.98	60,551	28,063.26	30,331.63
scpd1	60	8,301	13,734.22	13,585.83	82,581	137,534.78	137,611.67	75,296	125,392.69	127,811.67	70,222	116,936.09	124,061.67	61,935	103,124.93	108,736.67
scpd2	66	8,062	12,114.80	11,796.21	80,900	122,476.11	122,648.48	72,850	110,278.79	113,480.30	68,274	103,344.76	108,425.76	60,573	91,677.27	95,722.73
scpd3	72	8,297	11,424.26	11,295.14	82,511	114,498.15	114,009.03	76,658	106,369.32	110,672.22	69,456	96,366.26	101,744.44	62,426	86,603.15	92,198.61
scpd4	62	8,162	13,064.89	12,344.35	81,570	131,463.87	131,171.77	75,078	120,992.78	124,980.65	68,140	109,803.16	117,574.19	64,033	103,179.80	103,009.68
scpd5	61	8,105	13,186.45	13,095.08	82,008	134,340.00	134,600.00	76,733	125,691.08	128,432.79	68,896	112,843.61	122,583.61	60,042	98,329.82	104,753.28
scpe1	5	17	237.33	220.00	6	28.67	20.00	101	1,926.00	2,000.00	73	1,357.50	1,430.00	45	807.69	920.00
scpe2	5	16	227.33	220.00	6	20.00	20.00	107	2,036.52	2,020.00	71	1,316.30	1,240.00	49	887.20	960.00
scpe3	5	17	240.00	220.00	6	23.33	20.00	102	1,942.96	2,020.00	75	1,391.30	1,620.00	48	858.57	920.00
scpe4	5	17	246.67	240.00	7	34.67	20.00	100	1,905.93	1,980.00	73	1,356.00	1,620.00	41	712.00	820.00
scpe5	5	16	219.33	200.00	6	13.33	20.00	103	1,961.54	2,040.00	74	1,384.83	1,620.00	45	802.73	920.00
scpnre1	29	13,870	47,728.39	45,650.00	110,907	382,338.05	380,629.31	101,750	350,762.79	357,803.45	95,031	327,591.44	337,520.69	86,825	299,295.15	326,713.79
scpnre2	30	13,494	44,878.33	44,855.00	109,859	366,095.67	365,148.33	99,804	332,580.00	337,983.33	92,900	309,568.19	327,266.67	87,455	291,416.54	298,176.67
scpnre3	27	13,122	48,500.00	48,166.67	109,630	405,938.64	405,270.37	99,611	368,830.27	378,870.37	92,321	341,828.17	351,316.67	87,626	324,440.46	337,355.56
scpnre4	28	13,062	46,551.55	44,828.57	109,806	392,066.07	391,566.07	101,117	361,032.01	371,792.86	94,079	335,896.13	345,148.21	86,511	308,867.58	317,144.64
scpnre5	28	14,610	52,076.79	47,528.57	108,552	387,585.71	387,994.64	99,661	355,831.04	359,133.93	90,302	322,405.57	339,353.57	87,296	311,672.88	316,450.00
scpnrf1	14	13,864	98,930.24	97,035.71	108,140	772,325.00	769,296.43	98,916	706,440.82	707,614.29	91,617	654,309.62	674,171.43	86,236	615,870.63	650,171.43
scpnrf2	15	13,388	89,155.11	89,076.67	109,180	727,766.22	726,050.00	100,920	672,700.28	689,693.33	92,743	618,189.88	646,513.33	85,647	570,881.67	581,980.00
scpnrf3	14	13,855	98,861.90	96,232.14	110,060	786,045.00	786,746.43	99,431	710,120.54	727,939.29	93,309	666,394.51	677,257.14	84,591	604,122.79	599,250.00
scpnrf4	14	13,825	98,651.19	98,403.57	110,529	789,393.33	789,092.86	101,891	727,693.13	731,746.43	93,729	669,390.38	700,892.86	86,545	618,077.02	663,535.71
scpnrf5	13	13,859	106,510.51	105,280.77	112,077	862,033.33	861,896.15	99,194	762,933.92	768,469.23	94,708	728,420.60	760,915.38	87,480	672,822.79	702,200.00
scpnrg1	176	120,992	68,645.44	68,001.42	236,556	134,306.53	134,584.38	234,670	133,235.23	133,657.95	234,193	132,963.96	133,429.55	232,617	132,068.73	132,147.16
scpnrg2	154	111,172	72,089.39	71,425.32	236,045	153,176.04	153,385.39	235,380	152,744.01	153,463.96	232,952	151,167.69	152,313.96	230,825	149,786.66	149,705.19
scpnrg3	166	123,882	74,527.73	73,860.84	236,981	142,659.66	142,624.70	235,435	141,728.59	142,056.63	233,697	140,681.19	140,662.05	232,193	139,775.25	140,204.22
scpnrg4	168	122,636	72,897.78	68,901.79	239,246	142,308.49	142,430.65	237,427	141,225.41	141,786.31	236,898	140,910.77	141,391.07	233,172	138,692.78	138,707.74
scpnrg5	168	119,570	71,072.60	67,468.75	236,984	140,961.90	141,283.04	236,286	140,546.19	141,371.43	233,515	138,897.29	140,025.00	233,023	138,604.22	138,675.89
scpnrh1	63	126,184	200,192.49	183,626.98	236,953	376,016.56	376,395.24	235,341	373,456.44	373,642.86	233,188	370,039.39	371,013.49	230,641	365,996.32	366,069.84
scpnrh2	63	122,781	194,791.22	182,411.11	237,444	376,794.55	376,592.06	235,719	374,057.48	376,753.97	234,012	371,347.83	370,071.43	232,320	368,662.50	368,896.83
scpnrh3	59	129,597	219,556.38	208,127.12	235,709	399,406.61	399,915.25	234,078	396,642.55	398,116.95	232,771	394,427.50	395,117.80	230,455	390,502.10	390,837.29
scpnrh4	58	125,290	215,916.90	202,646.55	234,942	404,971.90	405,266.38	233,555	402,580.25	404,489.66	232,054	399,993.97	401,895.69	228,894	394,544.68	396,720.69
scpnrh5	55	124,218	225,750.55	206,576.36	235,845	428,708.85	429,398.18	233,380	424,227.64	426,714.55	233,021	423,574.74	424,316.36	231,729	421,226.29	422,810.91

Table D.2: The table presents the complete results for the SCP with Local Search. Each row represents a problem instance, and "BKS" stands for the best-known solution for that instance. $ACost$ is the average cost obtained within the independent executions of each method, $ADev_B$ is the average relative percentage deviations from the best solution known in the literature for that problem instance, and $MDev_B$ is the median relative percentage deviations from the best solution known in the literature.

Instance	BKS	Tuned-LS (1-hour)			Random-LS (1-hour)			Race-LS (1-hour)			Race-LS (2-hours)			Race-LS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
scp41	429	472	10.00	9.79	441	2.80	2.33	578	34.69	33.57	571	33.15	32.40	538	25.33	26.11
scp42	512	560	9.32	9.28	523	2.13	1.95	652	27.44	29.98	638	24.53	23.44	610	19.08	18.65
scp43	516	565	9.58	8.33	529	2.45	2.23	669	29.63	30.62	641	24.23	26.07	619	19.93	21.32
scp44	494	538	8.93	9.01	508	2.92	2.63	643	30.12	30.77	624	26.35	27.33	596	20.58	20.65
scp45	512	546	6.68	6.93	530	3.58	2.15	640	25.07	27.34	611	19.41	19.73	596	16.42	16.41
scp46	560	592	5.70	5.54	568	1.45	1.25	708	26.48	28.57	693	23.79	24.29	663	18.45	18.04
scp47	430	460	6.89	6.63	458	6.41	1.63	575	33.82	36.51	552	28.28	28.49	518	20.53	20.93
scp48	492	538	9.35	9.35	504	2.38	1.83	653	32.75	31.71	620	26.08	26.42	595	20.93	19.92
scp49	641	716	11.77	11.31	672	4.76	4.60	824	28.58	28.24	787	22.82	23.17	782	21.92	22.54
scp410	514	562	9.42	9.73	530	3.06	3.21	707	37.58	41.25	672	30.78	31.71	661	28.65	27.72
scp51	253	287	13.29	13.44	300	18.39	16.80	343	35.42	37.15	340	34.24	34.39	334	32.16	32.41
scp52	302	349	15.51	15.73	354	17.21	16.39	402	33.24	33.44	397	31.43	32.28	392	29.80	29.97
scp53	226	255	12.71	13.27	266	17.77	17.48	322	42.39	42.92	315	39.36	42.26	314	39.13	38.94
scp54	242	274	13.22	11.57	279	15.33	14.46	335	38.61	38.84	329	36.15	36.78	324	33.82	34.50
scp55	211	246	16.75	13.98	259	22.91	20.62	324	53.57	53.55	317	50.33	49.76	305	44.60	45.02
scp56	213	244	14.62	13.62	261	22.44	23.24	326	53.27	54.23	322	51.02	50.70	308	44.57	45.54
scp57	293	335	14.18	14.33	347	18.54	18.26	413	40.91	41.30	407	38.97	39.08	398	35.82	36.18
scp58	288	325	12.99	13.02	333	15.75	16.49	385	33.83	33.68	381	32.40	32.99	377	30.96	30.90
scp59	279	310	11.18	11.47	328	17.55	17.38	393	41.02	41.04	387	38.84	38.71	377	35.19	36.02
scp61	138	145	4.86	5.07	142	2.61	2.90	157	13.94	13.04	153	11.05	10.51	150	8.53	7.61
scp62	146	152	3.79	3.77	148	1.64	1.37	162	11.10	10.96	159	9.03	8.90	156	7.15	6.85
scp63	145	149	2.90	2.76	148	2.02	2.07	167	15.33	17.59	161	10.80	9.66	157	8.38	6.90
scp64	131	135	2.75	3.05	132	0.59	0.76	152	15.85	16.79	148	12.61	12.98	143	9.07	9.16
scp65	161	168	4.14	4.35	163	1.12	1.24	182	13.22	13.04	178	10.74	10.56	177	9.71	10.56
scp510	265	294	10.84	10.19	314	18.38	18.30	387	46.16	46.04	385	45.22	43.77	378	42.72	41.89
scpa1	253	335	32.46	34.19	338	33.62	34.58	367	45.19	46.25	362	43.27	44.27	358	41.54	41.90
scpa2	252	340	34.93	35.32	344	36.69	36.90	371	47.22	47.62	367	45.68	46.03	357	41.84	42.06
scpa3	232	307	32.47	31.47	309	33.28	34.05	335	44.33	45.26	332	43.23	43.53	328	41.20	40.95
scpa4	234	319	36.35	37.18	324	38.26	39.10	346	47.73	48.72	339	44.87	46.58	337	43.90	44.02
scpa5	236	317	34.48	35.81	324	37.20	36.65	354	49.86	50.85	346	46.59	47.46	342	45.10	45.34
scpb1	69	83	19.81	20.29	85	23.29	23.19	92	33.82	33.33	91	32.19	32.61	90	30.07	28.99
scpb2	76	92	20.48	21.05	94	23.64	23.68	98	29.25	29.61	97	27.19	27.63	95	25.23	25.66
scpb3	80	97	21.71	21.25	97	21.42	21.25	104	30.62	32.50	104	30.27	31.25	102	27.59	28.75
scpb4	79	93	17.72	17.72	94	19.62	18.99	102	29.36	30.38	100	26.58	27.85	97	22.72	22.78
scpb5	72	86	19.07	18.75	86	19.54	19.44	94	30.25	30.56	93	29.47	29.17	90	24.85	25.00
scpc1	227	340	49.97	50.22	340	49.68	51.10	351	54.74	55.51	346	52.32	53.52	337	48.31	48.90
scpc2	219	328	49.62	48.86	328	49.89	50.23	338	54.23	54.34	332	51.43	52.51	326	48.64	47.95

Instance	BKS	Tuned-LS (1-hour)			Random-LS (1-hour)			Race-LS (1-hour)			Race-LS (2-hours)			Race-LS (5-hours)		
		$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$	$ACost$	$ADev_B$	$MDev_B$
scpc3	243	357	46.75	47.53	359	47.67	48.35	367	51.18	51.23	364	49.93	51.44	357	46.82	47.33
scpc4	219	334	52.45	51.60	334	52.69	51.83	342	56.08	55.94	335	52.99	54.34	331	50.95	51.60
scpc5	215	324	50.67	50.93	325	51.19	51.40	334	55.16	56.74	331	53.99	54.19	323	50.44	51.63
scpd1	60	82	35.98	35.00	80	33.83	33.33	84	39.94	40.00	83	37.65	38.33	81	35.06	35.00
scpd2	66	87	31.31	30.30	87	31.52	30.30	90	36.74	36.36	88	32.75	33.33	86	29.97	30.30
scpd3	72	94	30.37	31.25	92	28.19	29.17	96	33.92	35.42	94	30.50	30.56	92	28.24	29.17
scpd4	62	83	34.62	33.87	83	34.25	33.87	86	38.13	38.71	85	37.45	37.10	83	33.58	33.87
scpd5	61	87	42.30	42.62	87	42.02	42.62	90	47.91	47.54	88	44.72	44.26	86	41.64	44.26
scpe1	5	10	100.00	100.00	10	99.33	100.00	10	100.00	100.00	10	100.00	100.00	10	99.29	100.00
scpe2	5	10	100.67	100.00	10	100.00	100.00	10	100.00	100.00	10	100.00	100.00	10	100.00	100.00
scpe3	5	10	100.67	100.00	10	96.00	100.00	10	100.00	100.00	10	100.00	100.00	10	100.00	100.00
scpe4	5	11	114.00	120.00	10	104.00	100.00	11	110.83	120.00	10	106.96	100.00	10	101.67	100.00
scpe5	5	10	97.93	100.00	9	88.67	80.00	10	96.43	100.00	10	91.54	100.00	9	86.92	80.00
scpnre1	29	34	18.62	18.97	34	18.05	17.24	35	20.83	20.69	34	18.14	17.24	34	16.97	17.24
scpnre2	30	36	21.11	20.00	36	20.56	20.00	36	20.80	20.00	36	19.75	20.00	36	19.05	20.00
scpnre3	27	33	21.48	22.22	33	22.59	22.22	33	23.59	25.93	33	21.63	22.22	32	18.90	18.52
scpnre4	28	34	20.71	21.43	34	20.71	21.43	34	22.89	25.00	34	21.18	21.43	33	19.09	17.86
scpnre5	28	34	22.98	25.00	35	23.57	25.00	35	25.13	25.00	34	22.02	21.43	34	20.13	21.43
scpnrf1	14	16	11.19	14.29	16	12.38	14.29	16	11.69	14.29	16	10.97	14.29	15	9.62	7.14
scpnrf2	15	16	8.67	6.67	16	6.67	6.67	16	7.88	6.67	16	7.47	6.67	16	4.67	6.67
scpnrf3	14	16	14.76	14.29	16	13.33	14.29	16	14.97	14.29	16	14.29	14.29	16	12.96	14.29
scpnrf4	14	16	11.19	14.29	16	12.14	14.29	16	12.03	14.29	16	10.71	10.71	15	9.61	7.14
scpnrf5	13	15	18.30	15.38	16	20.77	23.08	16	20.71	23.08	15	17.31	15.38	15	15.95	15.38
scpnrg1	176	290	64.72	65.62	286	62.42	61.93	288	63.71	63.64	285	61.80	62.78	278	58.19	58.52
scpnrg2	154	257	67.03	67.21	259	68.29	69.48	258	67.39	67.53	252	63.92	64.94	252	63.37	63.64
scpnrg3	166	271	63.03	63.25	270	62.67	62.65	271	63.38	63.55	271	63.42	63.86	263	58.27	58.73
scpnrg4	168	279	65.99	66.96	278	65.75	65.77	278	65.56	66.07	275	63.71	62.50	271	61.40	61.90
scpnrg5	168	274	63.06	63.39	272	61.94	62.50	272	61.79	63.10	267	59.19	59.52	266	58.04	58.04
scpnrh1	63	88	40.05	40.48	89	41.11	41.27	90	42.25	42.86	88	39.37	41.27	87	38.23	38.10
scpnrh2	63	90	42.06	42.86	89	41.69	41.27	89	42.03	42.86	89	41.50	41.27	86	37.15	36.51
scpnrh3	59	84	42.15	42.37	85	43.50	42.37	84	42.18	44.07	83	39.96	40.68	82	38.98	38.98
scpnrh4	58	82	42.13	43.10	82	42.18	42.24	82	41.24	41.38	80	38.53	37.93	80	37.93	37.93
scpnrh5	55	79	44.06	45.45	79	44.06	43.64	79	43.94	43.64	79	43.32	43.64	78	41.55	41.82