PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

BINARY GLOBAL-BEST HARMONY SEARCH ALGORITHM FOR SOLVING SET-COVERING PROBLEM

JUAN AGUSTÍN SALAS FERNÁNDEZ

THESIS TO APPLY FOR THE MASTER'S DEGREE
IN INFORMATIC ENGINEERING

JULY, 2016

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

BINARY GLOBAL-BEST HARMONY SEARCH ALGORITHM FOR SOLVING SET-COVERING PROBLEM

JUAN AGUSTÍN SALAS FERNÁNDEZ

Dr. Broderick Crawford Labrín
Master Thesis Advisor

PhD. Ricardo Soto de Giorgis
Master Thesis Co-Advisor

THESIS TO APPLY FOR THE MASTER'S DEGREE
IN INFORMATIC ENGINEERING

JULY, 2016

Table of Contents

List of Figures

List of Tables

Abstract

In this thesis, we propose a Binary Global-Best Harmony Search (BGBHS) Algorithm to solve different instances of the Set Covering Problem (SCP). The SCP belongs to the $\mathcal{NP}$*-hard* class problems where the number of alternate optimal solutions for the problem grows exponentially with the problem sizes. SCP has become a classic combinatorial optimization problem due to its many practical applications. In this document we consider applying BGBHS and Modified BGBHS supported in adaptive adjustment of probability parameter p in the Bernoulli trials when the harmonies are created. The different results presented in this thesis show that our algorithm is a good alternative at a low cost to solve the SCP.

Keywords: Binary Global-Best Harmony Search, Metaheuristic, Set Covering Problem.

1 Introduction

Every process is potentially optimizable. The vast majority of companies is involved in solving optimization problems every day [1]. Indeed, many challenging applications in science and industry can be modelled as optimization problems.

Metaheuristics are a branch of optimization in computer science that applie mathematics related to algorithms and computational complexity theory. Metaheuristics are raising a large interest in diverse technologies, industries, and services since they proved to be efficient algorithms in solving a wide range of complex real-life optimization problems in different domains [2].

The instantiation of a metaheuristic requires to choose among a set of different possible components and to assign specific values to all free parameters. Instantiations are defined, by the author, as a beginning configuration, and its relevance turns crucial on solving the SCP [3].

The SCP is a well-known mathematical problem, which tries to cover a set of needs at the lowest possible cost. SCP is very important in practice, as it has been used to model a large range of problems like scheduling, manufacturing, service planning, information retrieval, among others.

In an attempt to solve the SCP, a variation of Harmony Search metaheuristic in binary domain was used. This method has been tested using SCP instances from the OR-library [4] and this results have shown that the developed algorithm generates good solutions for each instance.

2 Main goal

The main goal of this work, is to solve the SCP using the Binary Global-Best Harmony Search metaheuristic algorithm. The performance of metaheuristic is validated against OR-library benchmarks resolutions.

2.1 Specific objectives

– To apply different adjustments to the parameters of initiation of the metaheuristic in experimental way to obtain better results.
– To perform a study of the parameters driven through the experiments to see if there is a possibility to optimize by creating, eliminating or modifying some of them.
– To repair the infeasible solutions through a repair operator.
– To compare and to analyze the results using different resolution strategies.
– To compare results with others metaheuristics.

3 Related terms

Operator: Unitary procedure for transforming information or implement the behavior of the algorithm.

Solution: Array of n columns containing a solution for a given problem. In binary case, posible values are 0 and 1.

Constrain: Conditions to be met to find a viable solution.

Benchmark: Optimal set of known problem instances to validate the propose algorithm.

Objective Function: Implements the mathematical expression representing the problem to solve. The guiding objective function is related to the goal to achieve.

Fitness: Value resulting by applying the objective function to a found solution.

Matrix of Costs: Consist of n columns vector containing the cost associated to each problem variable.

Optimal Value: Solution with the best fitness.

Domain: Set of feasible values for the variable.

Matrix A: Matrix containing the restrictions for the given problem.

RPD: Relative Percentage Deviation.

Harmony Memory: Memory space which includes the population of the solution vectors.

Harmony Memory Size: Defines the amount of harmonies that can be stored in HM.

Harmony Memory Consideration Rate (HMCR): In memory consideration, the value of decision variable x'_1 is randomly selected from the historical values, other decision variables, $(x'_2, x'_3, \ldots, x'_N)$ are sequentially selected in the same manner with probability where HMCR $\in$ (0,1).

Pitch Adjusting Rate (PAR): Each decision variable x'_i assigned a value by memory considerations is pitch adjusted with the probability of PAR where PAR $\in$ (0,1).

4 Theoretical framework

Optimization problems are encountered in many domains: BigData, engineering, logistics, and business. An optimization problem may be defined by the couple (S, f), where S represents the set of feasible solutions, and $f : S \implies \mathbb{R}$ the objective function to optimize. The objective function assigns to every solution $s \in S$ of the search space a real number indicating its worth. The objective function f allows to define a total order relation between any pair of solutions in the search space.
The global optimum is defined as a solution $s^* \in S$ and it has a better objective function than all solutions of the search space, i.e., $\forall s \in S$, $f(s^*) \leq f(s)$.

4.1 Metaheuristics and their classifications

The $meta$ and $heuristic$ are Greek words, $meta$ it means $higherlevel$ and $heuristics$ means $to\ find$, $to\ know$ or $to\ discover$. Metaheuristics are a set of intelligent strategies to enhance the efficiency of heuristic procedures.
A metaheuristic will be successful on a given optimizaction problem if it can provide a balance between exploration and exploitation. Exploration means to generate diverse solutions, associating them to different points of the search space on the global scale, while exploitation means to search in a local region by exploiting the information that a current good solution has been found in this region. Exploration is often directed by use of randomization, which enables an algorithm to have the ability to jump between different bounded spaces and gives to the algorithm the capability to search optimal solutions globally.

The different metaheuristic approaches can be characterized by different aspects concerning the search path they follow or how search space is exploited [5]. The majority of these algorithms has a stochastic behavior and mimics biological or physical processes. This presented classification was proposed by Beheshti et al. [6], and can be understood better as shown in the (Figure 1).

Nature-inspired vs. non-nature inspiration This class is based on the origin of algorithm. The majority of metaheuristics are nature-inspired algorithms such as Black Hole (BH) [7], Particle Swarm Optimization (PSO) [8] and Genetic Algorithms (GA) [9]. Also, some of them are non-nature-inspired algorithms like Iterated Local Search (ILS) [10] and TabuSearch (TS) [11].

Population-based vs. single-point search There is a certain group of metaheuristics, that can be classified by the number of solutions in their lifecycle or execution, such as Trajectory methods, which are algorithms working based on a single solution at any time (Figure 2). On the other hand, Population-based algorithms perform searches with multiple initial points in a parallel style. Examples of these metaheuristics can be: Harmony Search (HS) [12], GA [13] and PSO.

Dynamic vs. static objective function Another way of classifying metaheuristics, is by the way of utilizing the objective function. Some algorithms maintain the objective function intact throughout the execution cycle, while others modify the objective function according to information collected at runtime.
An example of the second case presented, is Guided Local Search (GLS) [14]. The idea behind this approach is to escape from local optima by changing the search landscape.

One vs. various neighborhood structures The majority of metaheuristic algorithms apply one single neighborhood structure. The fitness landscape topology remain unaltered thru the course of the algorithm while others, like Variable Neighborhood Search (VNS) [15], employ a set of neighborhood structures. This latter structure gives the possibility to diversify the search by swapping between different fitness landscapes.

Memory usage vs. memoryless methods One of the most interesting variables to classify a metaheuristic is undoubtedly use of memory. Short term usually is different from long term memory. The first kind usually keeps track of recently performed moves, or taken decisions. The second is usually an accumulation of synthetic parameters about the search.

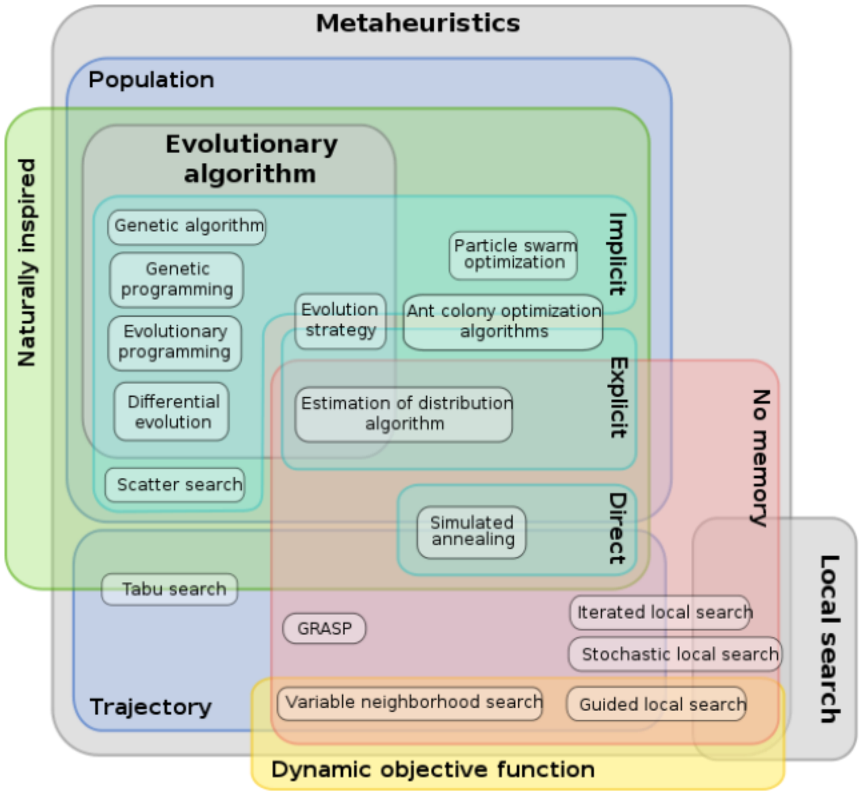


Fig. 1: Classification of metaheuristic

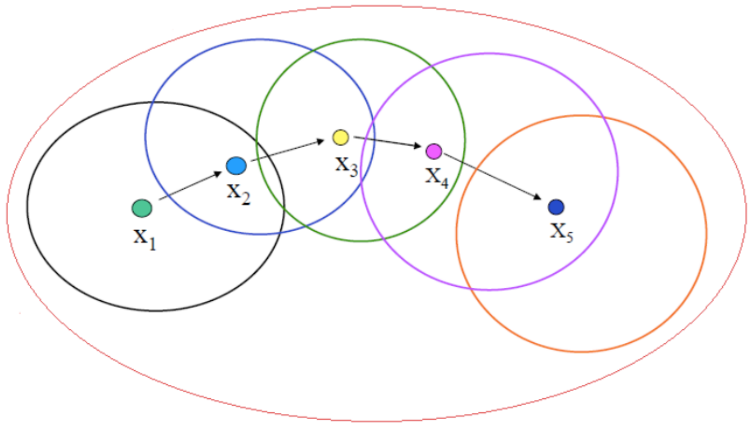


Fig. 2: Trajectory-based method

4.2 Development Framework

In an effort to maintain consistency in the structure of the proposed metaheuristic development, a well-defined model is followed, in order to structure the development steps of the proposed technique (Figure 5).

At first step, the problem is modeled and, based on its properties, application requirements are defined. In the particular case of this investigation the SCP is defined and validated against instances of benchmarck. The number of iterations has been considered, but runtime has not.

The second step basically corresponds to design the metaheuristic. In the case of this research, problems will be treated with variants of Harmony Search. That is to say not designed from the ground up, but certain elements are incorporated to improve the algorithm standard behavior. These elements are selected once known the nature of the problem to be solved and the metaheuristic to be used. Elements like the objective function to use and classification (Population based) are set by default.

The third step is to adopt a strategy of development of the technique or metaheuristic. In this research, it has been chosen to develop metaheruistics from scratch, thereby achieving maximum control of it. It was used for this purpose Python 2.7 as programming language and PyCharm as IDE.

The fourth step is handling optimization parameters. There are two types of parameter tuning: Online and Off line. In this research, management of both types is performed to achieve good results. Specifically improving the metaheuristic proposal is based on the dynamic variation of the parameter that generates solutions. The following sections will go into detail on this subject.

The fifth and final step is to design experiments, get results and compare improvements. Then perform changes in parameters or operators so as to keep improving convergence, achieving an optimum balance between exploration and exploitation. This framework adopts an iterative approach because when step five is completed, systems can return to step one, two or three to seek a better solution.

4.3 Set Covering Problem

4.3.1 SCP formulation The SCP is a well-known mathematical problem, which tries to cover a set of needs at the lowest possible cost. The SCP was included in the list of 21 $\mathcal{NP}$*-complet* problems of Karp [16]. There are many practical uses for this problem, such as: crew scheduling [17, 18], location of emergency facilities [19, 20], production planning in industry [21, 22, 23], vehicle routing [24, 25], ship scheduling [26, 27], network attack or defense [28], assembly line balancing [29, 30], traffic assignment in satellite communication systems [31, 32], simplifying boolean expressions [33], the calculation of bounds in integer programs [34], information retrieval [35], political districting [36], crew scheduling problems in airlines [37], among others. The SCP can be formulated as follows:

$$\text{Minimize} \quad Z = \sum_{j=1}^{n} c_j x_j \tag{1}$$

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i \in I \tag{2}$$

$$x_j \in \{0, 1\} \quad \forall j \in J \tag{3}$$

Let $A = (a_{ij})$ be a $m \times n$ 0-1 matrix with $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, n\}$ be the row and column sets respectively. We say that column j can be cover a row i if $a_{ij} = 1$. Where c_j is a nonnegative value that represents the cost of selecting the column j and x_j is a decision variable, it can be 1 if column j is selected or 0 otherwise. The objective is to find a minimum cost subset $S \subseteq J$, such that each row $i \in I$ is covered by at least one column $j \in S$. In the following section, we present a simple way to understand the SCP, through an example.

4.3.2 SCP sample solution Imagine that an ambulance station can meet the needs of an geographic zone. Similarly the ambulance station can cover all the needs of the nearby areas. For example, if a station is built in Zone 1 (Figure 3) ambulance station can meet the needs of neighboring areas, that is, it could also cover: Zone 1, Zone 2, Zone 3 and Zone 4. This can be appreciated in equation (5).
In this example, we must fulfill the need to cover the geographical areas defined in accordance with the restrictions.

The restriction of this case is that all areas must be covered by at least one ambulance station and the goal is to minimize the number of stations built, the cost of building a station is the same for all areas. The x_j variable represents the area j which is 1 if the ambulance station is built, and will be 0 if not. As above, it can be formulated as follows:

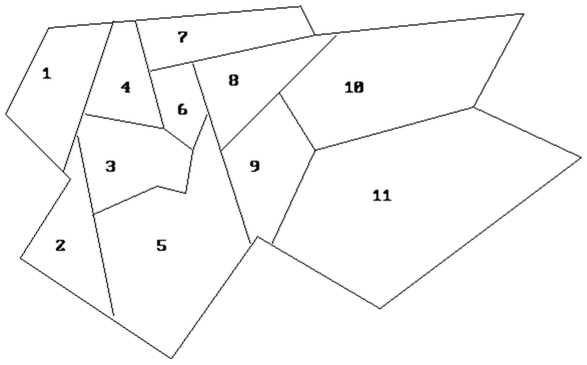


Fig. 3: Set Covering Problem example.

$$\text{Min} \quad c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 + c_5x_5 + c_6x_6 + c_7x_7 + c_8x_8 + c_9x_9 + c_{10}x_{10} + c_{11}x_{11} \tag{4}$$

Subject to:

$$x_1 + x_2 + x_3 + x_4 \geq 1 \tag{5}$$
$$x_1 + x_2 + x_3 + x_5 \geq 1 \tag{6}$$
$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 1 \tag{7}$$
$$x_1 + x_3 + x_4 + x_6 + x_7 \geq 1 \tag{8}$$
$$x_2 + x_3 + x_5 + x_6 + x_8 + x_9 \geq 1 \tag{9}$$
$$x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \geq 1 \tag{10}$$
$$x_4 + x_6 + x_7 + x_8 \geq 1 \tag{11}$$
$$x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \geq 1 \tag{12}$$
$$x_5 + x_8 + x_9 + x_{10} + x_{11} \geq 1 \tag{13}$$
$$x_8 + x_9 + x_{10} + x_{11} \geq 1 \tag{14}$$
$$x_9 + x_{10} + x_{11} \geq 1 \tag{15}$$

The first constraint (5) indicates that to cover zone 1, it is possible to locate a station in the same area or in the border. The following restriction is for zone 2 and so on. One possible optimal solution for this problem is to locate ambulance stations in zones 3, 8 and 9. That is, $x_3 = x_8 = x_9 = 1$ y $x_1 = x_2 = x_4 = x_5 = x_6 = x_7 = x_{10} = x_{11} = 0$. As shown in (Figure 4).

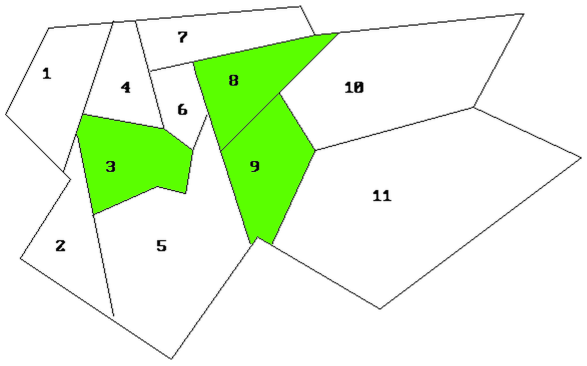


Fig. 4: Set Covering Problem solution.

The author propose solve the SCP, with a variation metaheuristic Harmony Search (HS) called Binary Global-Best Harmony Search to obtain satisfactory solutions within a reasonable time. HS mimics the process of musical improvisation, where musicians make adjustments in tone to achieve aesthetic harmony.

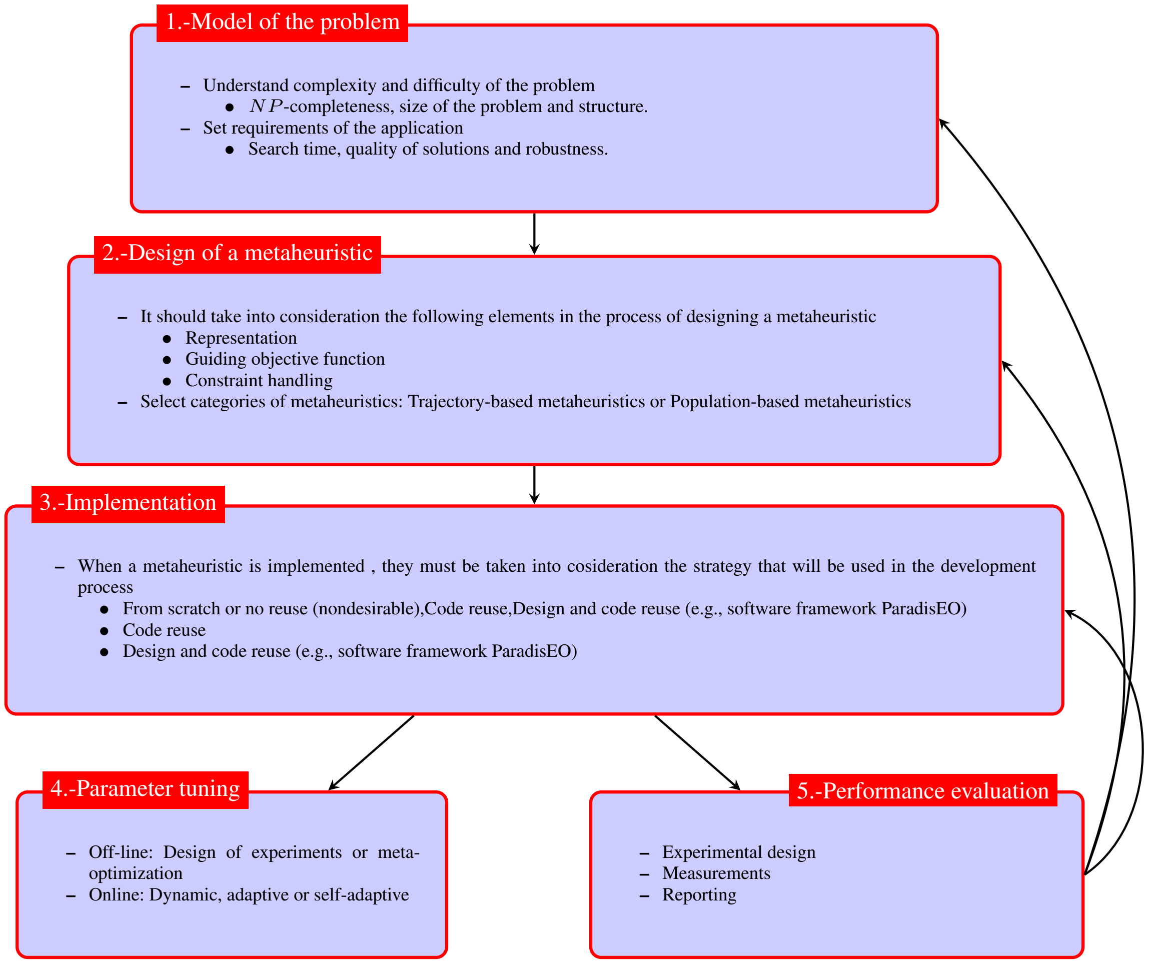

1.-Model of the problem
– Understand complexity and difficulty of the problem
• NP-completeness, size of the problem and structure.
– Set requirements of the application
• Search time, quality of solutions and robustness.
2.-Design of a metaheuristic
– It should take into consideration the following elements in the process of designing a metaheuristic
• Representation
• Guiding objective function
• Constraint handling
– Select categories of metaheuristics: Trajectory-based metaheuristics or Population-based metaheuristics
3.-Implementation
– When a metaheuristic is implemented , they must be taken into cosideration the strategy that will be used in the development process
• From scratch or no reuse (nondesirable),Code reuse,Design and code reuse (e.g., software framework ParadisEO)
• Code reuse
• Design and code reuse (e.g., software framework ParadisEO)
4.-Parameter tuning
– Off-line: Design of experiments or meta-optimization
– Online: Dynamic, adaptive or self-adaptive
5.-Performance evaluation
– Experimental design
– Measurements
– Reporting

Fig. 5: Guidelines for solving SCP.

4.4 Traditional Harmony Search and some variants

4.4.1 Harmony Search Metaheuristic is a population-based metaheuristic algorithm inspired from the musical process of searching for a perfect state of Harmony or Aesthetic Quality of a Harmony (AQH). The HS was proposed by Z. W. Geem et al. [38]. In other words, the main idea of the HS metaheuristic is to mimic the process performed by musicians when they try to play a beautiful harmony.

For the purpose of properly understand what is looking like a good solution in this metaheuristic, we must know the meaning of AQH. The AQH in an instrument is essentially determined by its pitch (or frequency), sound quality, and amplitude (or loudness). The sound quality is mainly determined by the harmonic content that is in turn, determined by the waveforms or modulations of the sound signal. However, the harmonics that it can generate will largely depend on the pitch or frequency range of the particular instrument [39]. Different notes have different frequencies. For example, the note A has a fundamental frequency $f_0 = 440Hz$. The fundamental frequency of each note can be seen in (Table 1).

Musical Note	Frequency
C	261,625565 Hz
D	293,664768 Hz
E	329,627557 Hz
F	349,228231 Hz
G	391,995436 Hz
A	440,000000 Hz
B	493,883301 Hz

Table 1: HS - Musical Notes

Given the above, it is established that a good harmony has good AQH. The pitch of each musical instrument determines the AQH, just as the fitness function values determines the quality of solution.
In the music improvisation process, all musicians (Table 2) sound pitches (Table 3) within possible range together to make one harmony (Table 4), If all pitches make a good harmony, each musician stores in his memory that experience and possibility of making a good harmony (Table 5) in increased next time. The same thing in optimization: the initial solution is generated randomly from decision variables within the possible range, If the objective function values of these decision variables is good to make promising solution, then the possibility to make a good solution is increased next time.
In this document, the researcher set its focus on studying the classical SCP problem, given by equation (1), where we want minimize the cost of solution.

The traditional HS metaheuristic has five steps [40], which will be reviewed below.

	Each musician represents a decision variable, according to the example shown in sub-section 4.3.2, there would be 11 musicians, since there are 11 decision variables $x_1, \ldots, x_{11}$.

Table 2: HS components - Musician

	The pitch range of the instrument represents the range of values that can take a decision variable. Given the nature of the SCP, the possible values are $\{0, 1\}$.

Table 3: HS components - Pitch range

	Musical harmony at a certain time, corresponds to a solution at a certain iteration.

Table 4: HS components - Solution

	Aesthetics audience, judges whether harmony is good or not. In the problem it refers to the objective function.

Table 5: HS components - Aesthetics audience

Step 1: Initialize the problem and algorithm parameters.
The optimization problem is defined as shown in equation (1) where the goal is to minimize Z subjec to equations (2) and (3). x_{iL} and x_{iU} are the lower and upper bounds for decision variables. The required parameters to solve the optimization problem, that is, Harmony Memory Size (HMS) or number of solution vectors in the harmony memory, Harmony Memory Consideration Rate (HMCR) which determines the rate of selecting the value from the memory and Pitch Adjusting Rate (PAR) which determines the probability of local improvement and number of improvisations

(NI) are given when the metaheuristic begins.

Step 2: Initialize the harmony memory.
An initial HM is filled with a population of Harmony Memory Size (HMS). We can formally say the following equation:

$$x_i^d(t+1) = x_{iL}^d(t) + rand()(x_{iU}^d(t) - x_{iL}^d(t)),\ \forall i \in \{1, 2, \ldots, n\}. \tag{16}$$

Where rand() is a random from a uniform distribution of $[0, 1]$.

Step 3: Improvise a new harmony.
Harmonies are generated randomly and the details of the procedure to improvise harmony $x_i^{'}$ can be given in Algorithm 1. According to the process, vectors are generated as follows $(x^1, \ldots, x^{HMS})$. Vectors results , make up a matrix such as that shown in (Figure 6):

$$HM = \begin{bmatrix} x_1^1 & \ldots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_1^{HMS} & \ldots & x_n^{HMS} \end{bmatrix}$$

Fig. 6: Harmony Memory Matrix

Algorithm 1: Generating new harmony for traditional HS

```
1 for $i \leftarrow 1$ to $n$ do
2     if rand()≤HMCR then
3         $x_i^{'}$ = $x_i^j$ ($j = 1, 2, \ldots$,HMS) //memory consideration
4         if rand() $\leq$ PAR then $x_i^{'}$ = $x_i^{'} \pm r(bw)$ ;
5         //bw means band width
6     else
7         $x_i^{'}$ = $x_{iL}$ + rand() $(x_{iU} - x_{iL})$ //random selection
```

Step 4: Update harmony memory.
If the new generated harmony $x^{'}$ = $(x_1^{'}, x_2^{'}, \ldots, x_n^{'})$ has a better fitness than the worst one x_{worst} in HM, then replace the worst harmony with the new one as shown in the algorithm 2; otherwise , go to the next step.

Algorithm 2: Replace worst harmony procedure

```
1 if fitness($x^{'}$ ) > fitness($x_{worst}$) then
2     $x_{worst} = x^{'}$
```

Step 5: Check the stop criteria.
If the stopping criterion (e.g. maximum number of iterations NI) is satisfied, computation is terminated. Otherwise, step 3 is repeated.

4.4.2 Global-Best Harmony Search Metaheuristic

To further improve the convergence performance of HS and overcome some shortcomings of HS, a variant of HS, called Global-Best Harmony Search (GHS), was proposed by Omran and Mahdavi [41]. The initial population in GHS is generated randomly using a Bernoulli process. Then, a greedy vector is generated x_0 based on a profit ratio, if x_0 is best than x_{worst} in HM then x_{worst} is replaced by x_0. The GHS dynamically updates parameter PAR according to equation (17):

$$PAR(t) = PAR_{min} - \frac{PAR_{max} - PAR_{min}}{NI}t \tag{17}$$

where $PAR(t)$ represents the pitch adjusting rate at generation t, PAR_{min} and PAR_{max} are the minimum and maximum adjusting rate, respectively. The parameter t is the iterative variable, and parameter NI is the number of improvisations.
The parameter HMCR with a larger value can be helpful to accelerate the convergence speed, while the parameter HMCR with a smaller value can help to get out of local minima at the end of search [42]. The variation of the parameter depending on the iterations can be reviewed in equation (18).

$$HMCR(t) = HMCR_{min} - \frac{HMCR_{max} - HMCR_{min}}{NI}t \tag{18}$$

where $HMCR_{min}$ and $HMCR_{max}$ represent the lower and upper bounds of $HMCR$, respectively.

At first, musicians most likely choose a perfect state of a harmony from their memory or harmony memory during the process of improvising a new harmony. Next, they may select a pitch from the current harmony memory randomly and then they would perform a fine tune operation, that is, pitch adjustment, for the chosen pitch to improve the effectiveness of music. For the SCP, the states of a pitch just include $\{0, 1\}$, based on this, a genetic mutation is suitable for pitch adjustment [43]. The process of improvising a new harmony can be given in Algorithm 3.

Algorithm 3: Generating a new harmony in GHS

```
1 for $j \leftarrow 1$ to $n$ do
2     if $rand(0,1) \leq HMCR(t)$ then
3         $x_j^{'} = x_{best_j}$ //memory consideration
4     else
5         Generate a random integer number $a \in \{1, 2, \ldots, HMS\}$
6         $x_j^{'} = x_{best_a}$ //pitch adjustment for the pitch chosen randomly
7         if $rand(0,1) \leq PAR(t)$ then
8             $x_j^{'} = |x_j^{'} - 1|$ // discrete genetic mutation
```

The Selection Mechanism is a process in which a new generated harmony vector x_{new} is compared with x_{best} in the first place, and if x_{new} is better than x_{best}, replace x_{best} with x_{new}; otherwise x_{new} is compared with x_{worst} in HM and a greedy selection is applied between $\{x_{new}, x_{worst}\}$. As a consequence, GHS not only speeds up the convergence speed but also avoids being trapped in a local optimum.

Algorithm 4: Selection Mechanism procedure

```
1 if fitness($x_{new}$) > fitness($x_{best}$) then
2     $x_{best} = x_{new}$
3 else
4     if fitness($x_{new}$) > fitness($x_{worst}$) then
5         $x_{worst} = x_{new}$
```

5 Algorithm implementation

5.1 Implementation architecture

The implementation of the algorithm was developed using the Python programming language which allows work quickly and integrate systems more effectively. As data persistence, a MySql relational database was used, although log files were also used. All of the above was mounted on an instance of the free Amazon layer, specifically EC2 - Ubuntu GNU/Linux. As a source code control system, GIT was used, which allowed to maintain a control of local source code for both the developed algorithm and the documentation associated to this thesis. Constant replicas were performed against a GIT cloud repository, specifically GITHUB.

5.2 Binary Global-Best Harmony Search Metaheuristic

The parameters presented below were the result of exhaustive tests and multiple executions that allowed to probe beyond the random components a correct behavior for metaheuristics, that is to say, presenting a higher degree of effectiveness compared to other metaheuristics.

5.2.1 Improvising a New Harmony The process begins with the generation of an initial population of harmonies that are vectors composed of binary digits d, until the amount defined by the HMS. All harmonies are repaired after being created through two-phase repair operator (Algorithm 5).

Then in a repetition of the process that is called improvisation new harmonies are created. The number of improvisations is denoted by NI. As the harmonies are created, the two-phase operator ADD and DROP is applied to make the harmonies feasible. Once the harmonies are repaired and meet the constraints defined in the Matrix A for the SCP, the best and worst harmony, denoted by x_{best} and x_{worst}, respectively, are stored. Additionally a greedy selection mechanism are integrated into the BGBHS. The greedy operation is based on the idea that the item with higher profit density ratio should be used first. And the profit density ratio can be calculated by the equation (19):

$$\mu_j = \frac{1}{c_j} \tag{19}$$

The detail of the behavior of the metaheuristics can be reviewed in the flow chart (Figure 7) .

The HS is good at identifying the high performance regions of the solution space in a reasonable time, but poor at performing local search [42]. Namely, there is an unbalance between the exploration and the exploitation.

To solve the problem of local exploitation, we propose a variable parameter for the Bernoulli probability p in initial population, which starts from one and decreases with each iteration, tending to zero, presenting a behavior according to the equation (20):

$$p(t) = \frac{t}{t+1} \tag{20}$$

The graph of the function can be seen in the figure 8.

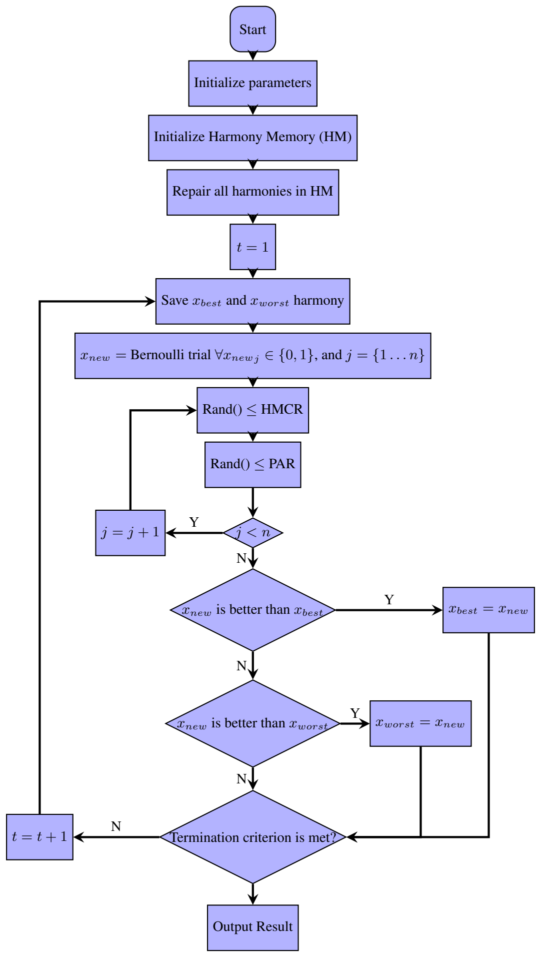

Start
Initialize parameters
Initialize Harmony Memory (HM)
Repair all harmonies in HM
$t = 1$
Save x_{best} and x_{worst} harmony
x_{new} = Bernoulli trial $\forall x_{new\,j} \in \{0,1\}$, and $j = \{1 \dots n\}$
Rand() $\leq$ HMCR
Rand() $\leq$ PAR
$j < n$
Y
$j = j + 1$
N
x_{new} is better than x_{best}
Y
$x_{best} = x_{new}$
N
x_{new} is better than x_{worst}
Y
$x_{worst} = x_{new}$
N
Termination criterion is met?
N
$t = t + 1$
Output Result

Fig. 7: Flowchart of BGBHS algorithm.

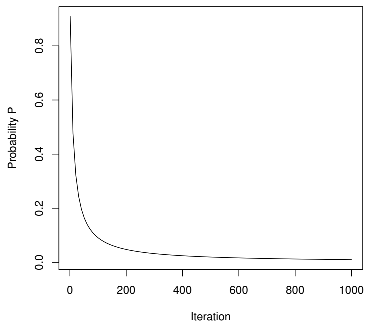


Fig. 8: p value over iteration

This procedure will allow solutions in a first instance to have more variables activated and as the iterations pass, the probability that a variable is activated will be closed to zero, in this way the initial population will have a wider range of solutions allowing to improve the balance between exploitation and exploration.
The initial population in BGBHS is generated randomly using a Bernoulli process. In probability and statistics, a Bernoulli process is a finite or infinite sequence of binary random variables x_1, x_2, x_3,…,x_n $\forall x \in \{0,1\}$ and

$i = \{1, \ldots, n\}$, so it is a discrete-time stochastic process that takes only two values, success (equation 21) or failure (equation 22), given the probability p.

$$P(x_i = 1) = P(\text{success at the } i\text{-th trial}) = p \tag{21}$$

$$P(x_i = 0) = P(\text{failure at the } i\text{-th trial}) = 1 - p \tag{22}$$

Specifically, for each decision variable of an initial harmony vector, a number within is generated randomly. If the value of the number is less than p, the corresponding variable takes 0; otherwise it takes 1. In this way, a set of HMS harmonies will be generated randomly. All the harmonies generated by this process are probably not feasible, only through the repair operator ADD and DROP become feasible. This means that for each harmony generated, the operator must be applied to comply with the restrictions.

5.2.2 Two-Phase Repair Operator: ADD and DROP New generated harmony vector needs to be repaired under two cases. One is that the harmony vector violates the constraints. The other corresponds to inactivate the most expensive columns, maintaining the feasibility of the solution. Hence, the repair operator consists of two phases. The first phase, called ADD, is responsible for repairing a harmony vector violating the constraint. The second phase, named DROP, is applied to remove any redundant column such that by removing it from the solution, the solution still remains feasible. The detail of the operation can be reviewed in the Algorithm 5.

Algorithm 5: Repair operator ADD and DROP

```
1: //ADD Phase
2: $M \leftarrow 1, 2, \ldots, m$
3: $A_i \sum_{j=1}^{n} a_{ij} x_j, i \in M$
4: for $j \leftarrow 1$ to $n$ do
5:     if $x_j = 0$ and $\exists i \in M, A_i < 1$ then
6:         $x_j \leftarrow 1$
7:         $A_i \leftarrow A_i + a_{ij}$
8:     end if
9: end for
10: //DROP Phase
11: for $j \leftarrow n$ to $1$ do
12:     if $x_j = 1$ and $\exists i \in M, A_i - a_{ij} \geq 1$ then
13:         $x_j \leftarrow 0$
14:         $A_i \leftarrow A_i - a_{ij}$
15:     end if
16: end for
```

6 Proposed improvements

In order to test the correct execution of the metaheuristic, a total of 30 executions were performed for each instance of the benchmark, both with the p parameter of Bernoulli fixed, and with the parameter p variable according to the function previously shown in equation (20).

Previously an exhaustive work was done, in order to define the most correct parameters for each instance, but on this occasion, only the tests were executed with 10 iterations until an optimal adjustment of the parameters was achieved. A particular issue is that the execution time of the tests ranged from 10 minutes to 3 hours, between instances. Given that motive, the executions were a smaller amount.

In the multiple experiments performed, throughout this work, it was verified that it is not convenient to modify the repair operators. For this reason this process was used as is. The ditail can be seen in the aforementioned biography.

The main improvement proposal presented in this paper has to do with the modification of the initial population generation at the beginning of metaheuristics, which aims to improve the balance between exploitation and exploration. Mainly, the exploration is improved, making the initial solution population present a much wider variation, starting with solutions where there are many activated columns $x_i = 1$ and ending with solutions that have many inactivated variables $x_i = 0$ in HM, has shown in matrix (23).

$$HM = \begin{bmatrix} 1,1,1,1,1,1,\ldots,1,0,1,1 \\ 1,1,0,1,1,1,\ldots,1,0,1,1 \\ 1,1,0,0,1,1,\ldots,1,0,1,1 \\ \vdots \\ 0,0,1,0,0,0,\ldots,1,0,0,0 \end{bmatrix} \tag{23}$$

Obviously each solution must be treated by the repair operator. For the first solutions, which have a greater number of variables activated, a more intense use of DROP functionality will be made, until the solution is made feasible. On the other hand, the last solutions will have a smaller number of columns activated, so it is necessary to make intensive use of the ADD functionality of the repair operator.

Independent of the balance that occurs, it was demonstrated throughout the experiments that there is a greater variation in the population which has a positive impact on the convergence of the algorithm.

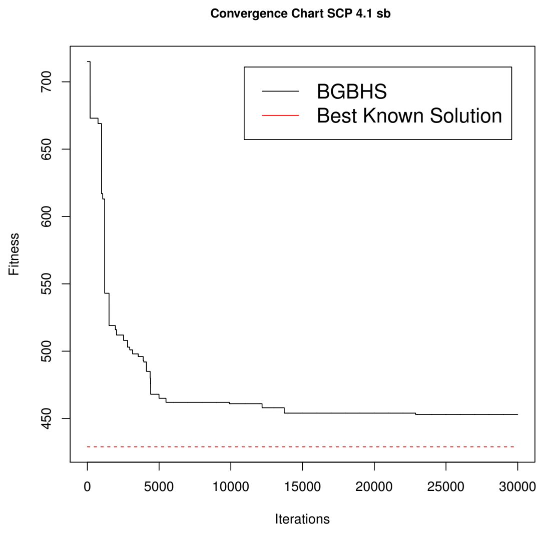
7 Experimental results

Resultados

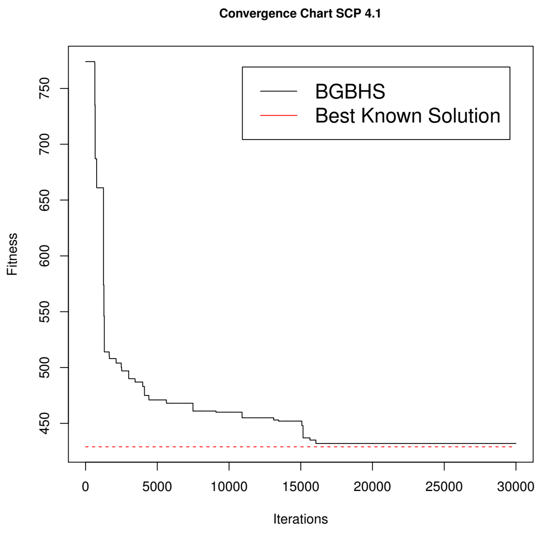
8 Analisys and conclusion

Conclusión

9 Appendix



(a)



(b)

Fig. 9: Parameter p fixed versus adaptive p parameter.

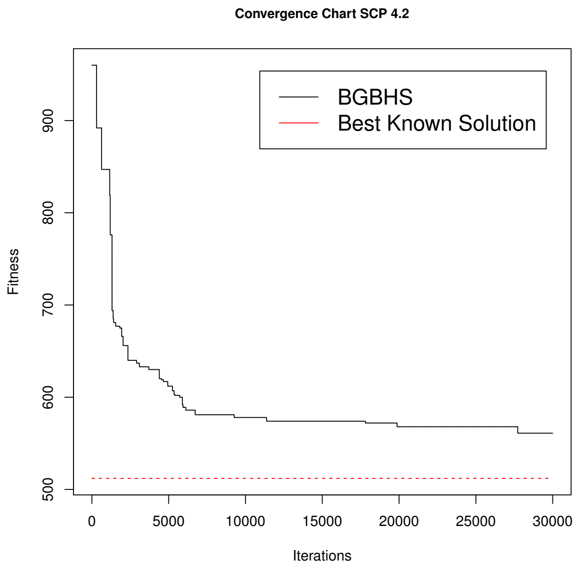


Fig. 10: Instance 4.2.

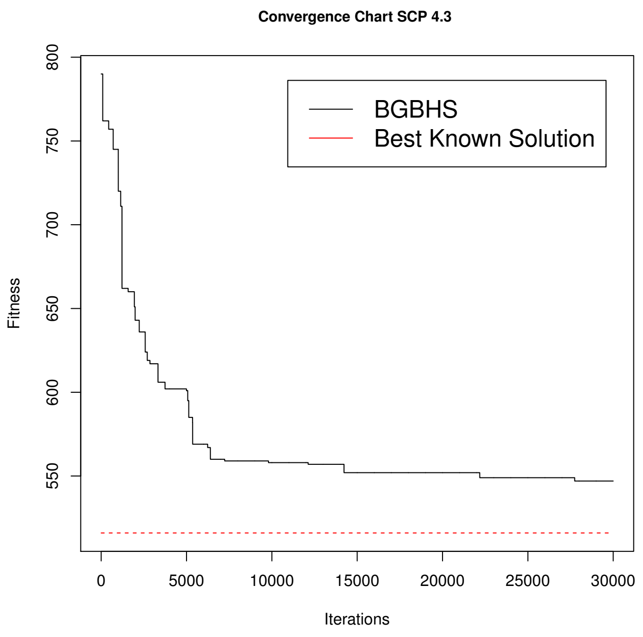


Fig. 11: Instance 4.3.

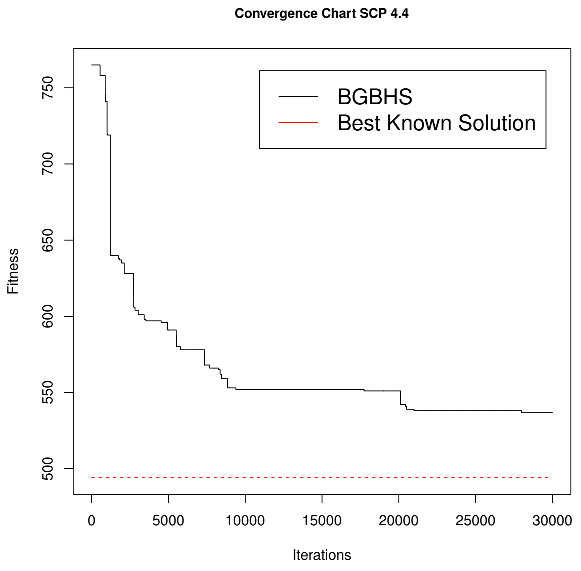


Fig. 12: Instance 4.4.

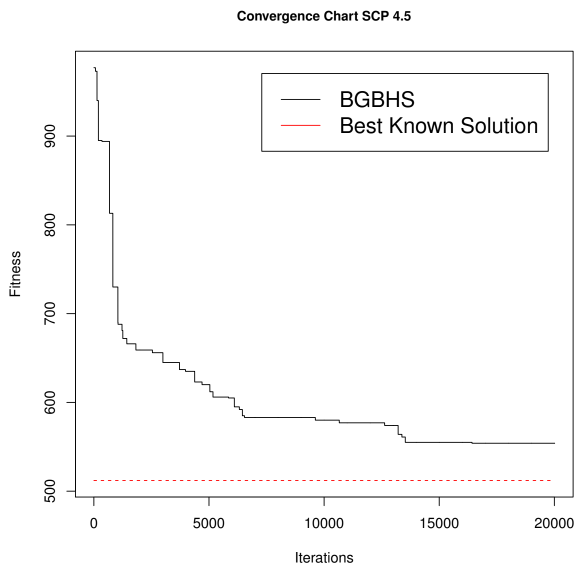


Fig. 13: Instance 4.5.

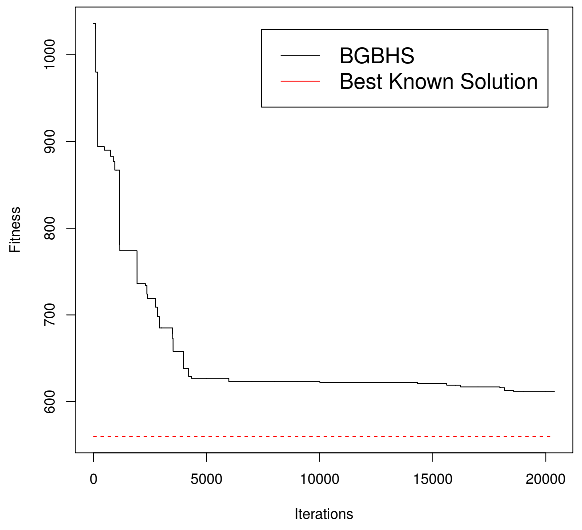


Fig. 14: Instance 4.6.

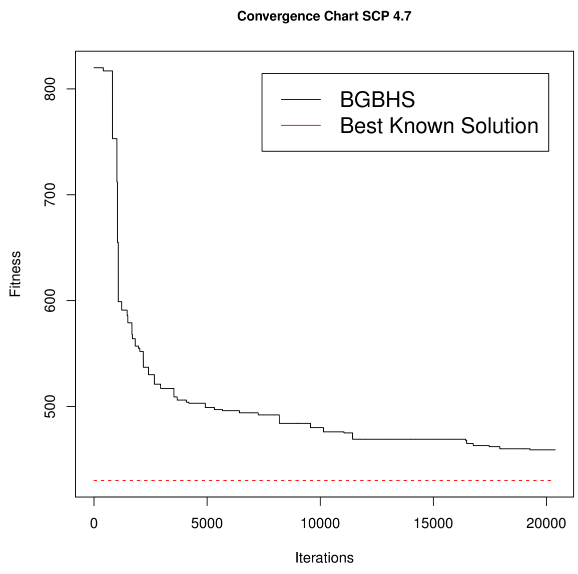


Fig. 15: Instance 4.7.

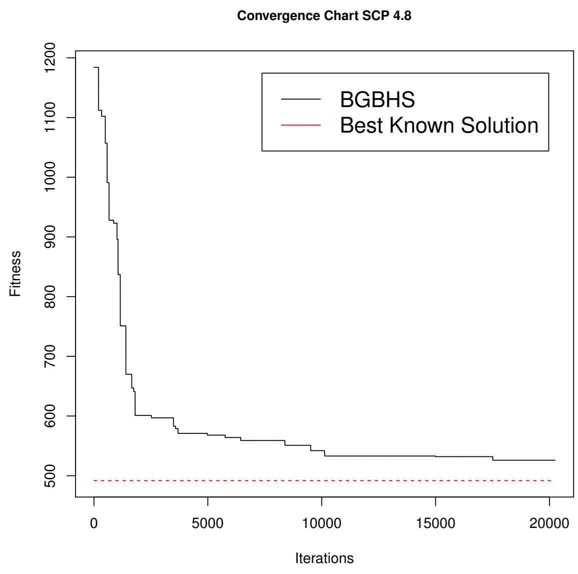


Fig. 16: Instance 4.8.

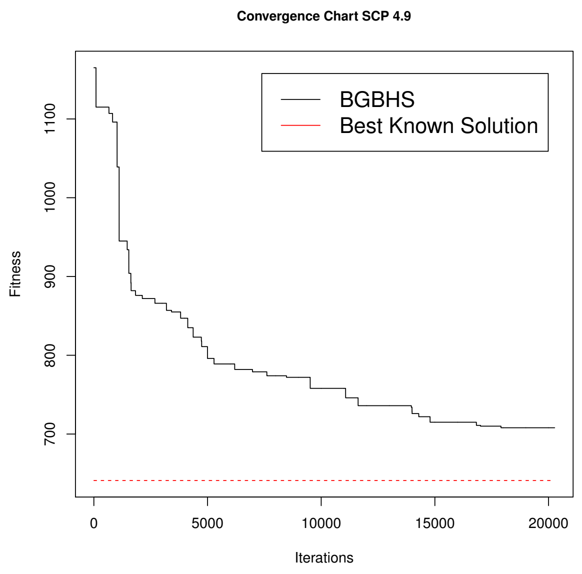


Fig. 17: Instance 4.9.

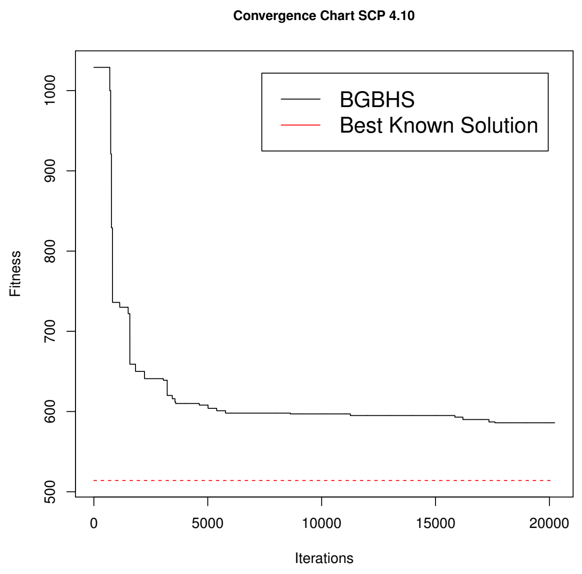
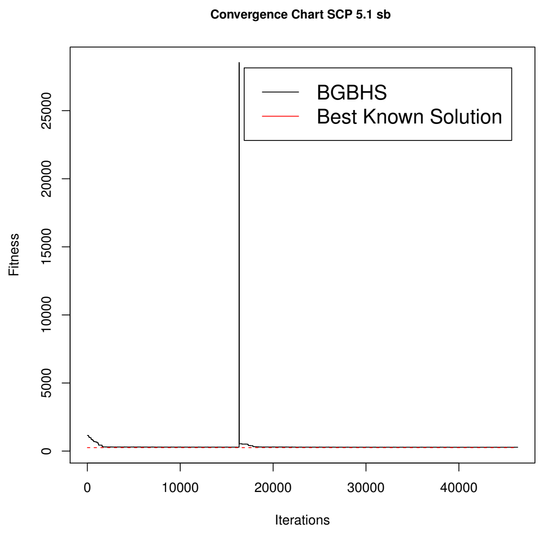
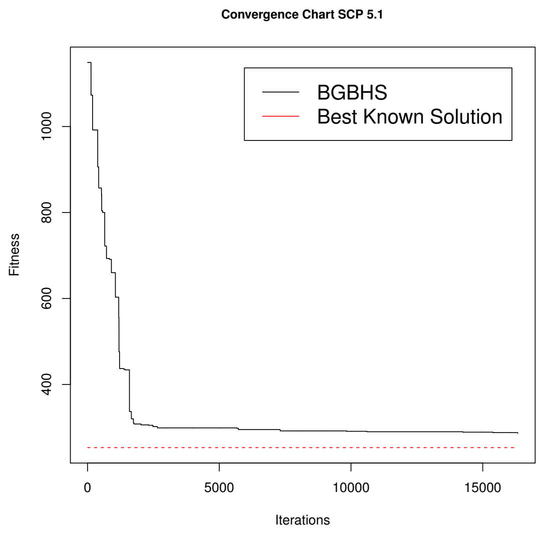


Fig. 18: Instance 4.10.



(a)



(b)

Fig. 19: Parameter p fixed versus adaptive p parameter.

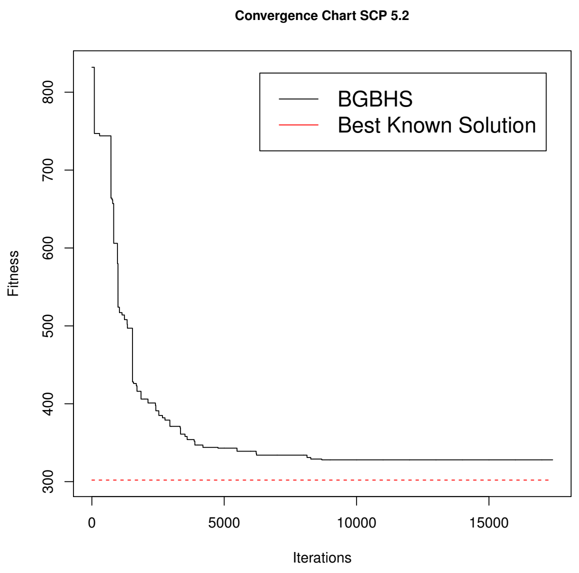


Fig. 20: Instance 5.2.

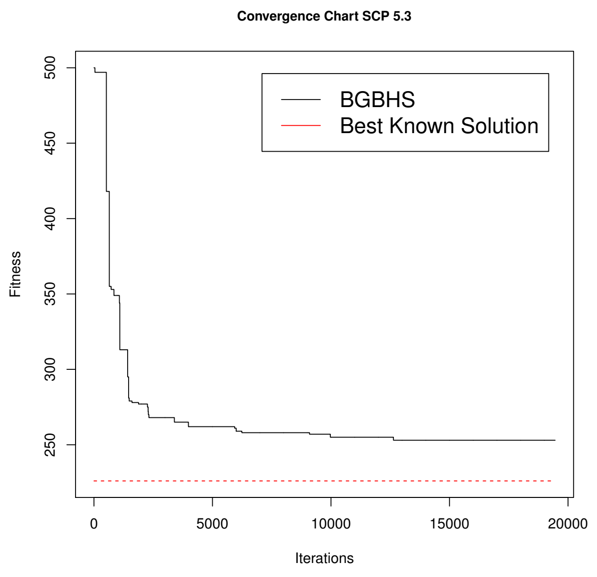


Fig. 21: Instance 5.3.

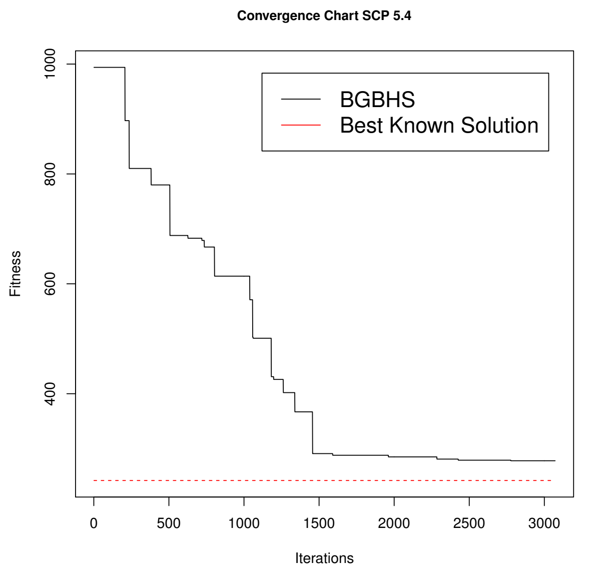


Fig. 22: Instance 5.4.

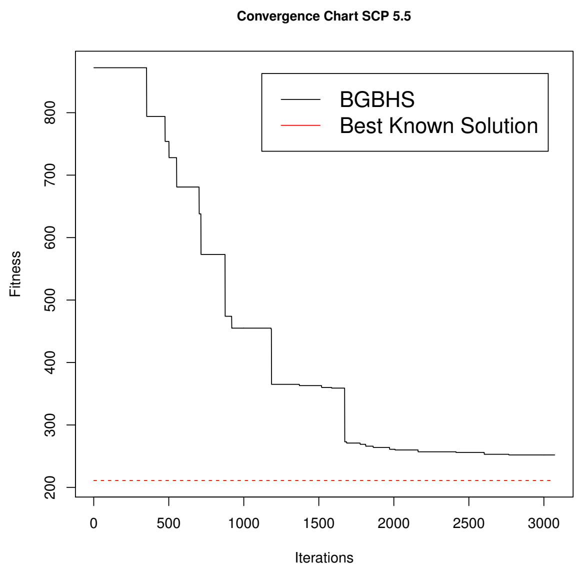


Fig. 23: Instance 5.5.

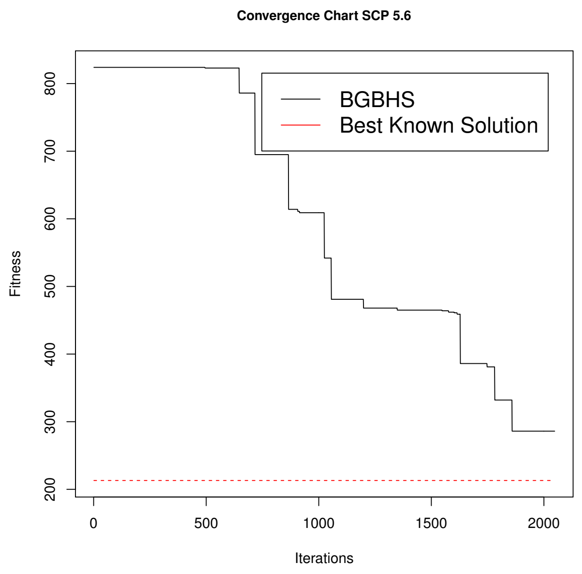


Fig. 24: Instance 5.6.

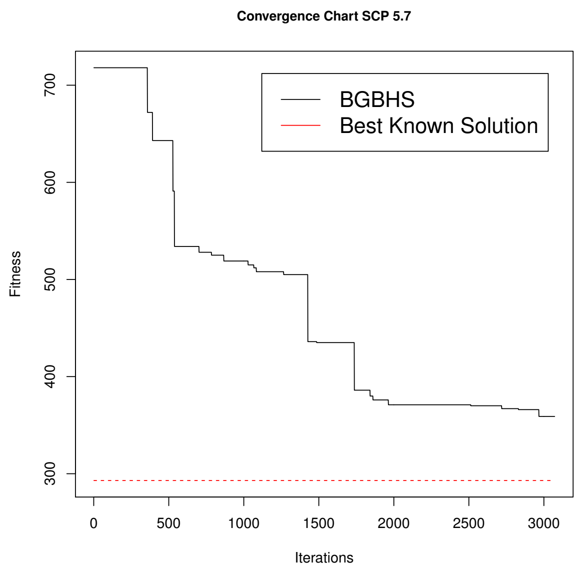


Fig. 25: Instance 5.7.

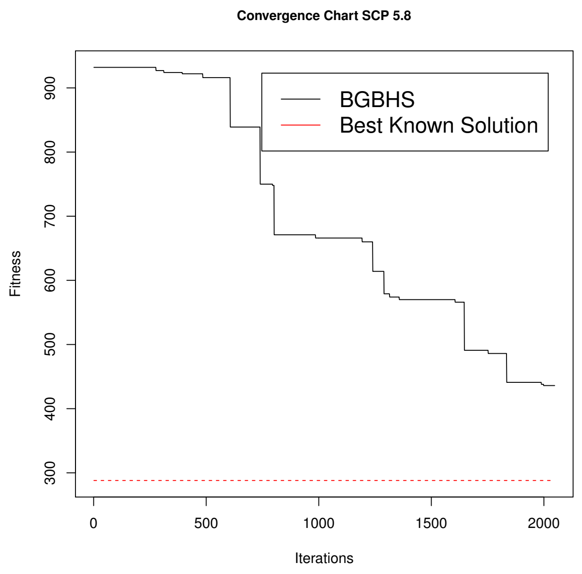


Fig. 26: Instance 5.8.

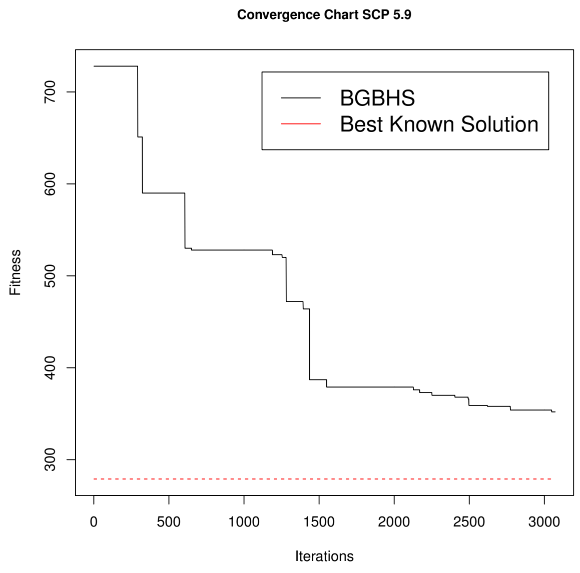


Fig. 27: Instance 5.9.

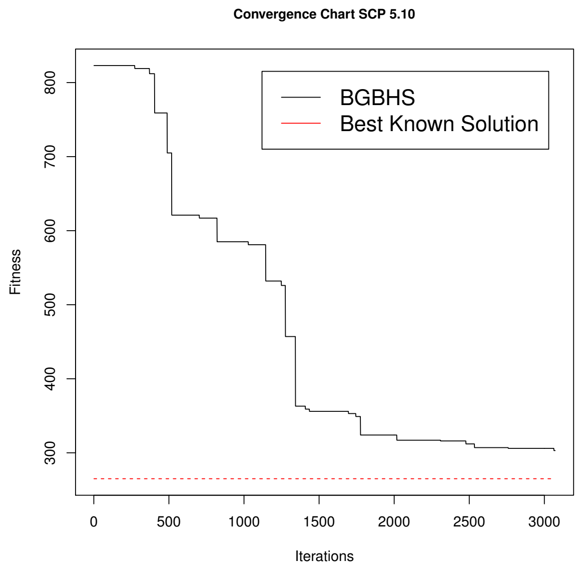


Fig. 28: Instance 5.10.

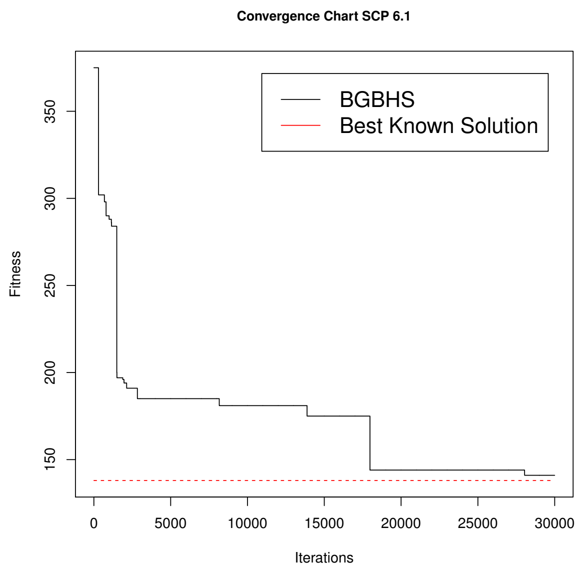


Fig. 29: Instance 6.1.

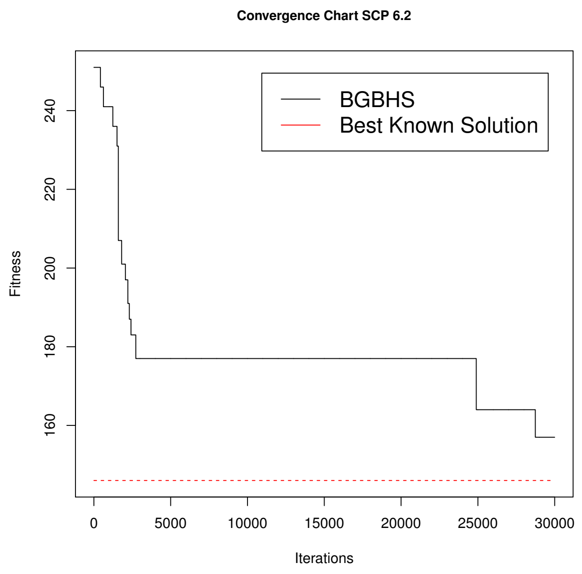


Fig. 30: Instance 6.2.

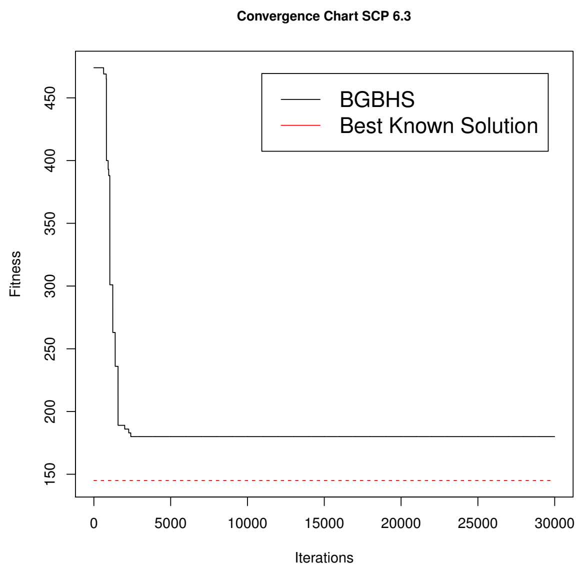


Fig. 31: Instance 6.3.

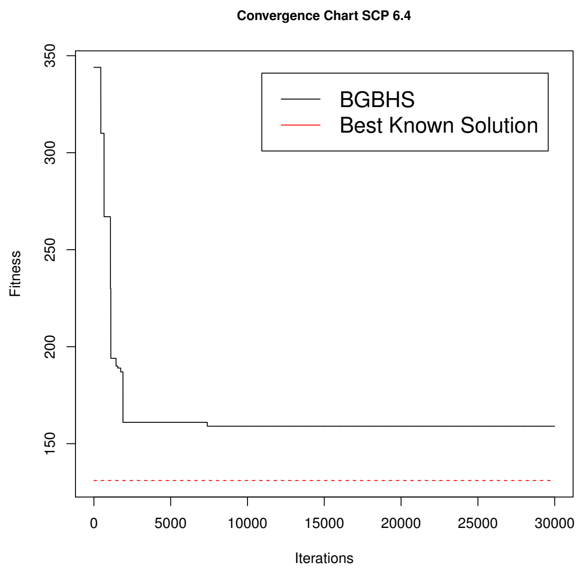


Fig. 32: Instance 6.4.

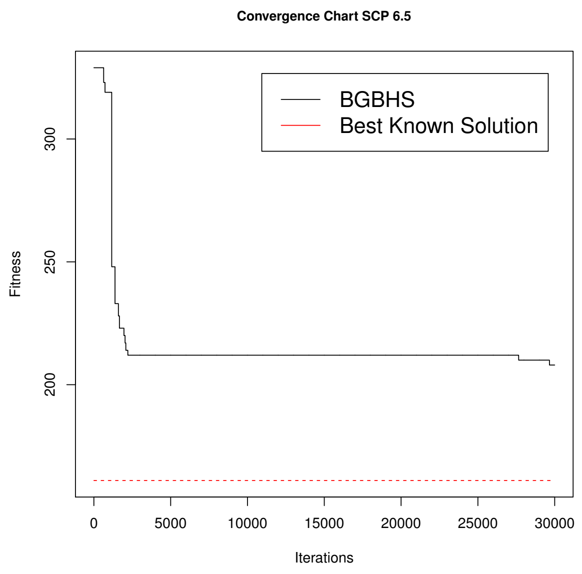


Fig. 33: Instance 6.5.

Fig. 34: Instance A.1.

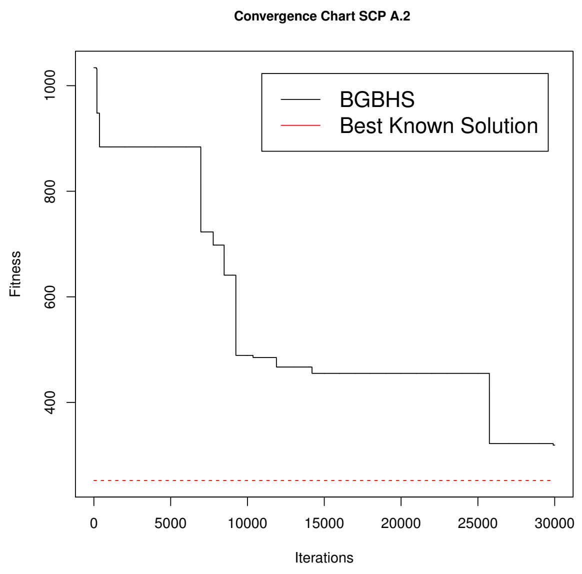


Fig. 35: Instance A.2.

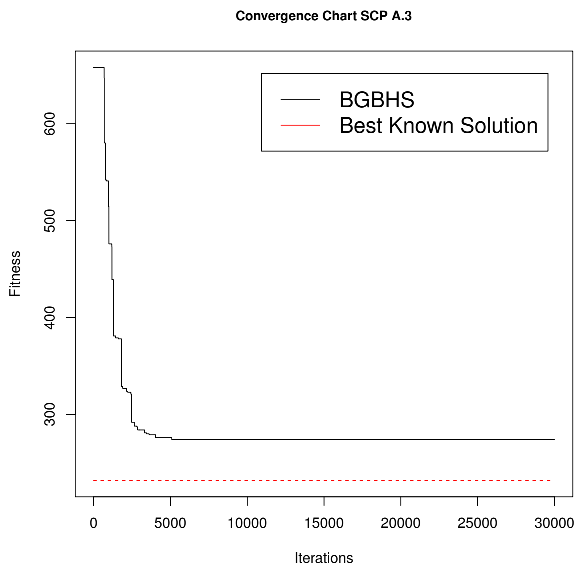


Fig. 36: Instance A.3.

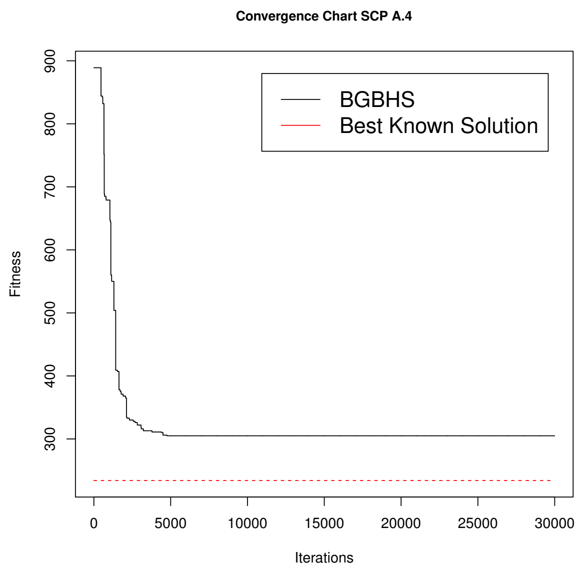


Fig. 37: Instance A.4.

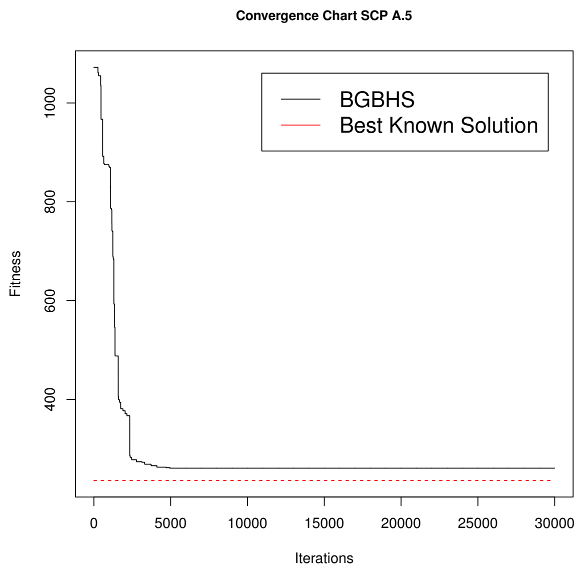


Fig. 38: Instance A.5.

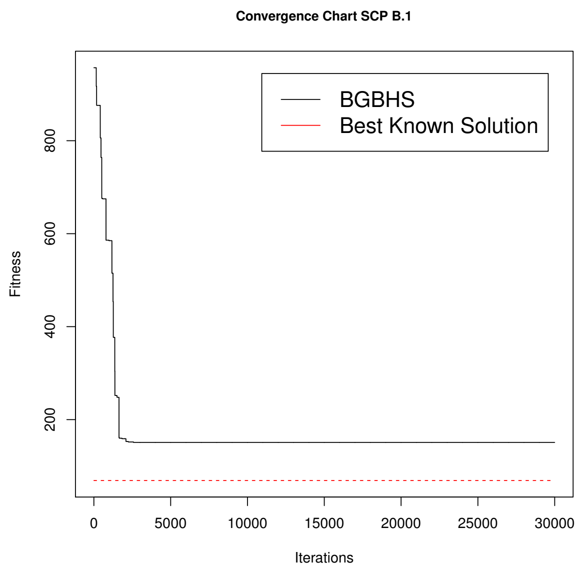


Fig. 39: Instance B.1.

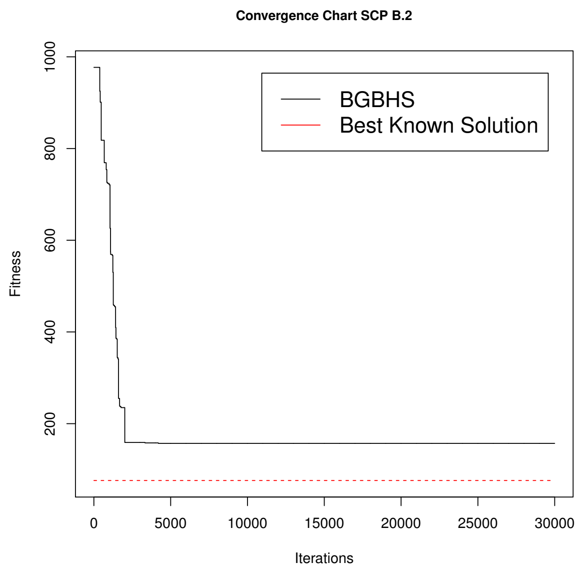


Fig. 40: Instance B.2.

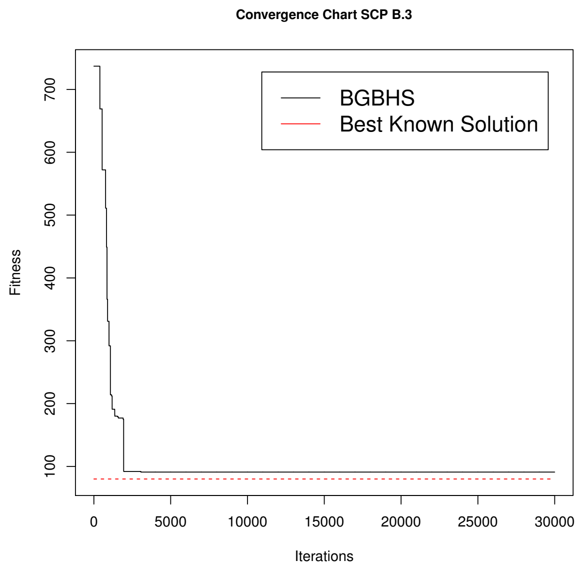


Fig. 41: Instance B.3.

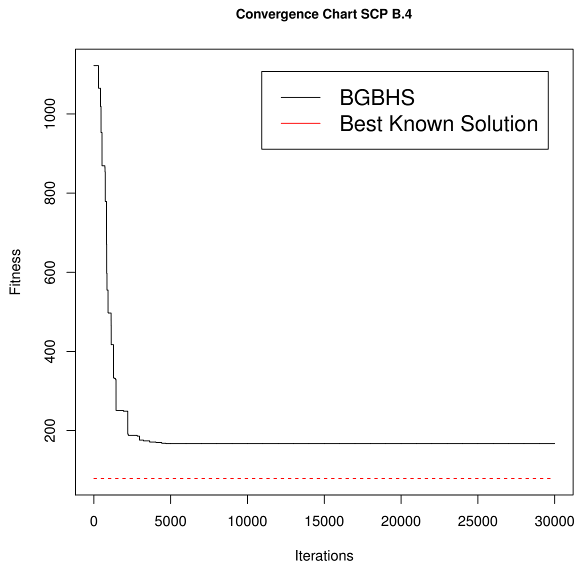


Fig. 42: Instance B.4.

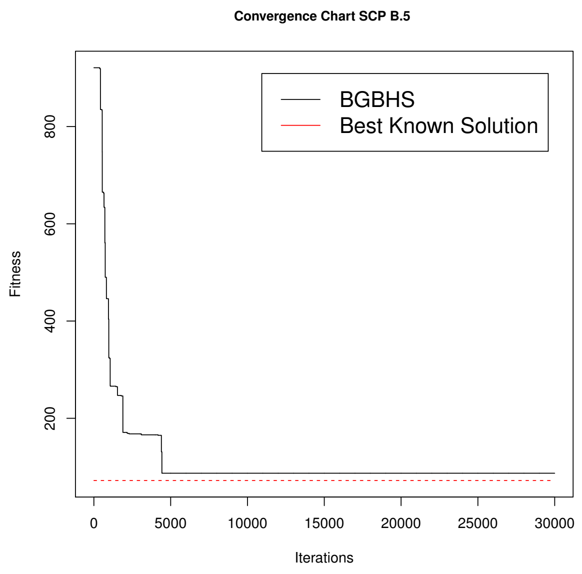


Fig. 43: Instance B.5.

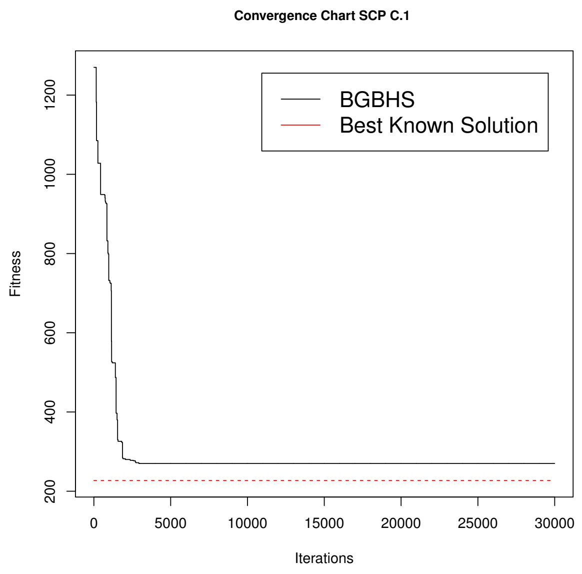


Fig. 44: Instance C.1.

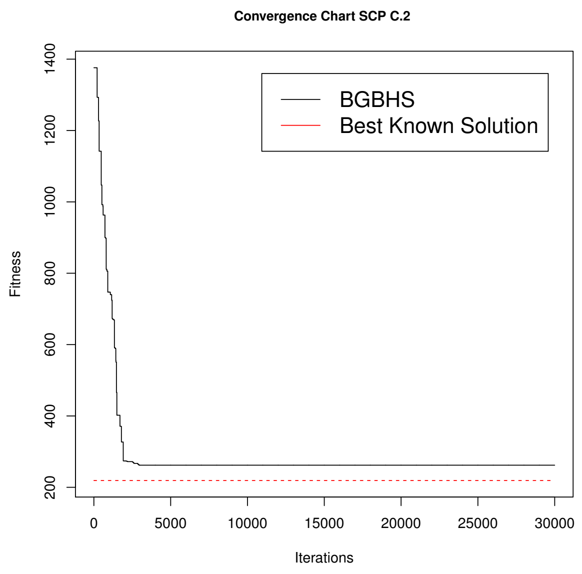


Fig. 45: Instance C.2.

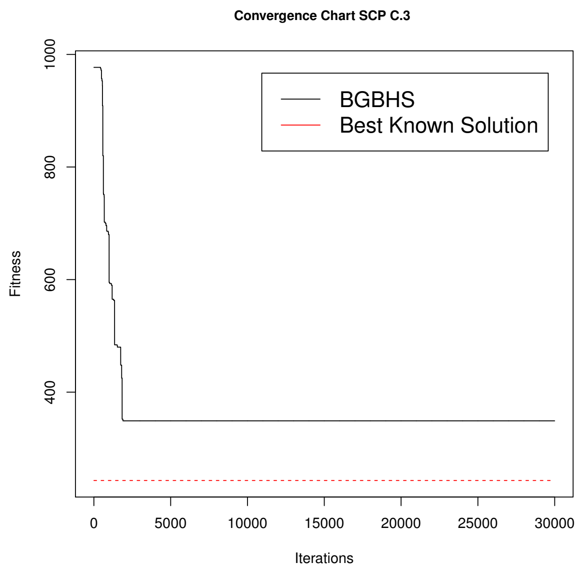


Fig. 46: Instance C.3.

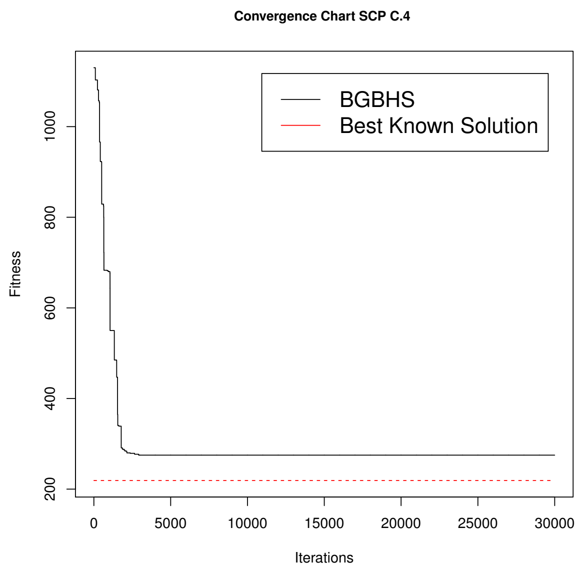


Fig. 47: Instance C.4.

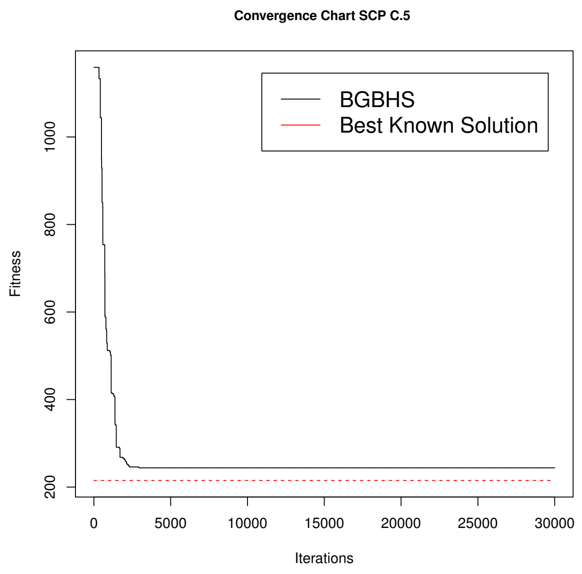


Fig. 48: Instance C.5.

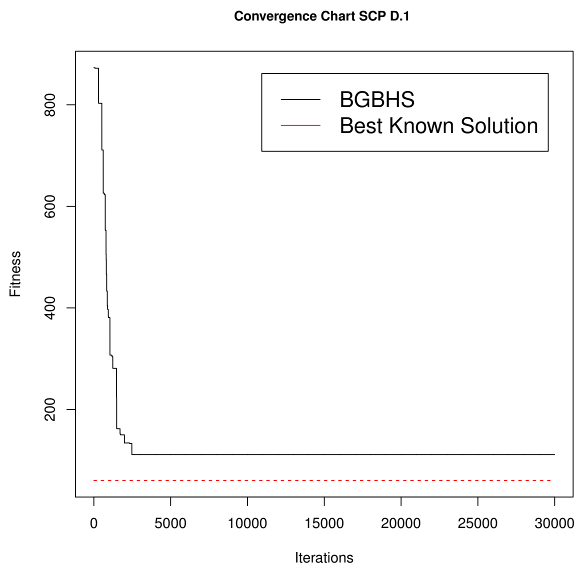


Fig. 49: Instance D.1.

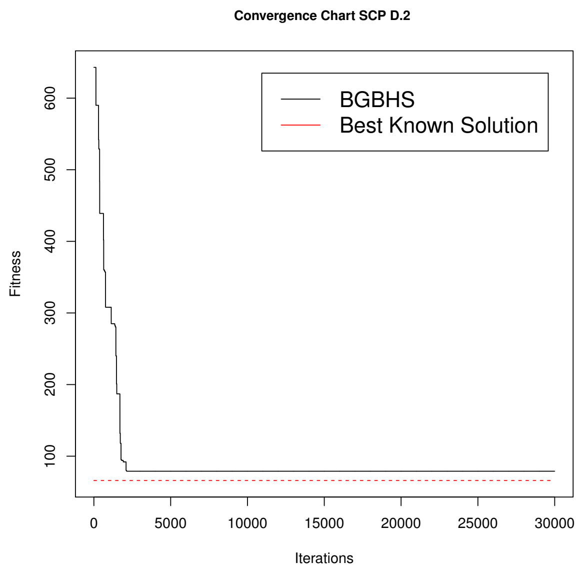


Fig. 50: Instance D.2.

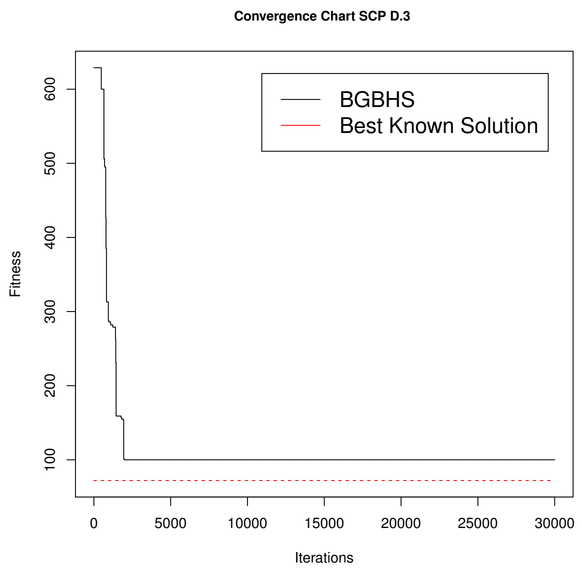


Fig. 51: Instance D.3.

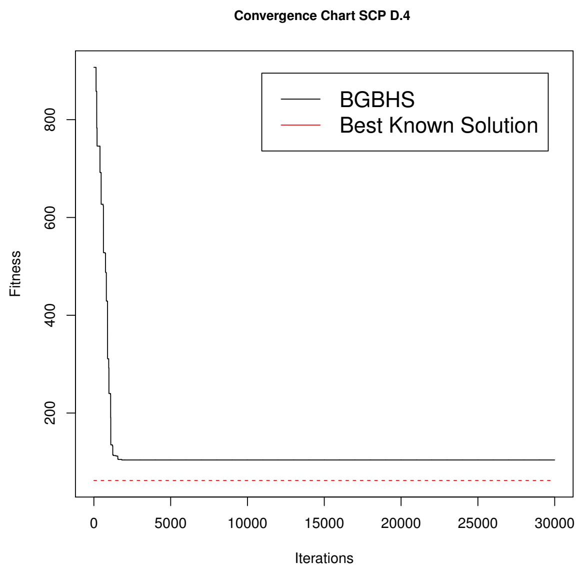


Fig. 52: Instance D.4.

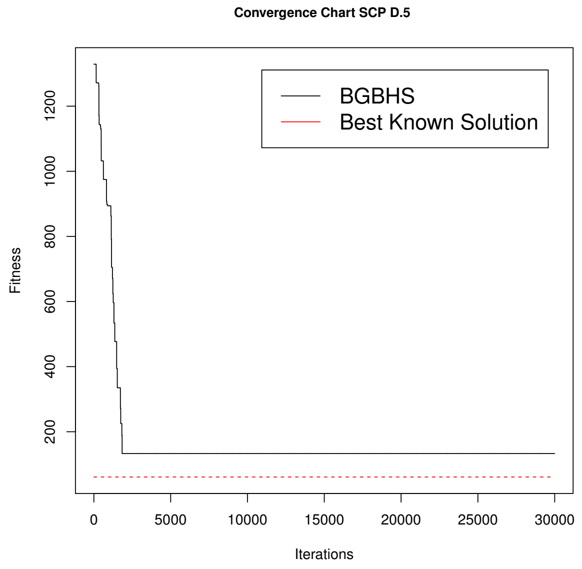


Fig. 53: Instance D.5.

9.1 Instance 4.1

Method	Optimum	Min	Max	Avg	RPD
BGBHS	429	534			24.475524475524477
BGBHS	429	553			28.904428904428904
BGBHS	429	533			24.242424242424242
BGBHS	429	600			39.86013986013986
BGBHS	429	438			2.0979020979020979
BGBHS	429	460			7.2261072261072261
BGBHS	429	455			6.0606060606060606
BGBHS	429	470			9.5571095571095572
BGBHS	429	533			24.242424242424242
BGBHS	429	532			24.009324009324008
BGBHS IMPROVED	429	451			5.1282051282051286
BGBHS IMPROVED	429	440			2.5641025641025643
BGBHS IMPROVED	429	460			7.2261072261072261
BGBHS IMPROVED	429	445			3.7296037296037294
BGBHS IMPROVED	429	470			9.5571095571095572
BGBHS IMPROVED	429	430			0.23310023310023309
BGBHS IMPROVED	429	430			0.23310023310023309
BGBHS IMPROVED	429	432			0.69930069930069927
BGBHS IMPROVED	429	435			1.3986013986013985
BGBHS IMPROVED	429	430			0.23310023310023309

Table 6: Instance 4.1

9.2 Instance 4.2

Method	Optimum	Min	Max	Avg	RPD
BGBHS	512	632			23.4375
BGBHS	512	662			29.296875
BGBHS	512	660			28.90625
BGBHS	512	602			17.578125
BGBHS	512	641			25.1953125
BGBHS	512	644			25.78125
BGBHS	512	642			25.390625
BGBHS	512	647			26.3671875
BGBHS	512	641			25.1953125
BGBHS	512	623			21.6796875
BGBHS IMPROVED	512	550			7.421875
BGBHS IMPROVED	512	560			9.375
BGBHS IMPROVED	512	563			9.9609375
BGBHS IMPROVED	512	520			1.5625
BGBHS IMPROVED	512	523			2.1484375
BGBHS IMPROVED	512	555			8.3984375
BGBHS IMPROVED	512	524			2.34375
BGBHS IMPROVED	512	518			1.171875
BGBHS IMPROVED	512	519			1.3671875
BGBHS IMPROVED	512	520			1.5625

Table 7: Instance 4.2

9.3 Instance 4.3

Method	Optimum	Min	Max	Avg	RPD
BGBHS	516				-100
BGBHS	516				-100
BGBHS	516				-100
BGBHS	516				-100
BGBHS	516				-100
BGBHS	516				-100
BGBHS	516				-100
BGBHS	516				-100
BGBHS	516				-100
BGBHS	516				-100
BGBHS IMPROVED	516	613			18.7984496124031
BGBHS IMPROVED	516				-100
BGBHS IMPROVED	516				-100
BGBHS IMPROVED	516				-100
BGBHS IMPROVED	516				-100
BGBHS IMPROVED	516				-100
BGBHS IMPROVED	516				-100
BGBHS IMPROVED	516				-100
BGBHS IMPROVED	516				-100
BGBHS IMPROVED	516				-100

Table 8: Instance 4.3

9.4 Instance 4.4

Method	Optimum	Min	Max	Avg	RPD
BGBHS	494				-100
BGBHS	494				-100
BGBHS	494				-100
BGBHS	494				-100
BGBHS	494				-100
BGBHS	494				-100
BGBHS	494				-100
BGBHS	494				-100
BGBHS	494				-100
BGBHS	494				-100
BGBHS IMPROVED	494				-100
BGBHS IMPROVED	494				-100
BGBHS IMPROVED	494				-100
BGBHS IMPROVED	494				-100
BGBHS IMPROVED	494				-100
BGBHS IMPROVED	494				-100
BGBHS IMPROVED	494				-100
BGBHS IMPROVED	494				-100
BGBHS IMPROVED	494				-100
BGBHS IMPROVED	494				-100

Table 9: Instance 4.4

9.5 Instance 4.5

Method	Optimum	Min	Max	Avg	RPD
BGBHS	512				-100
BGBHS	512				-100
BGBHS	512				-100
BGBHS	512				-100
BGBHS	512				-100
BGBHS	512				-100
BGBHS	512				-100
BGBHS	512				-100
BGBHS	512				-100
BGBHS	512				-100
BGBHS IMPROVED	512				-100
BGBHS IMPROVED	512				-100
BGBHS IMPROVED	512				-100
BGBHS IMPROVED	512				-100
BGBHS IMPROVED	512				-100
BGBHS IMPROVED	512				-100
BGBHS IMPROVED	512				-100
BGBHS IMPROVED	512				-100
BGBHS IMPROVED	512				-100
BGBHS IMPROVED	512				-100

Table 10: Instance 4.5

9.6 Instance 4.6

Method	Optimum	Min	Max	Avg	RPD
BGBHS	560				-100
BGBHS	560				-100
BGBHS	560				-100
BGBHS	560				-100
BGBHS	560				-100
BGBHS	560				-100
BGBHS	560				-100
BGBHS	560				-100
BGBHS	560				-100
BGBHS	560				-100
BGBHS IMPROVED	560				-100
BGBHS IMPROVED	560				-100
BGBHS IMPROVED	560				-100
BGBHS IMPROVED	560				-100
BGBHS IMPROVED	560				-100
BGBHS IMPROVED	560				-100
BGBHS IMPROVED	560				-100
BGBHS IMPROVED	560				-100
BGBHS IMPROVED	560				-100
BGBHS IMPROVED	560				-100

Table 11: Instance 4.6

9.7 Instance 4.7

Method	Optimum	Min	Max	Avg	RPD
BGBHS	430				-100
BGBHS	430				-100
BGBHS	430				-100
BGBHS	430				-100
BGBHS	430				-100
BGBHS	430				-100
BGBHS	430				-100
BGBHS	430				-100
BGBHS	430				-100
BGBHS	430				-100
BGBHS IMPROVED	430				-100
BGBHS IMPROVED	430				-100
BGBHS IMPROVED	430				-100
BGBHS IMPROVED	430				-100
BGBHS IMPROVED	430				-100
BGBHS IMPROVED	430				-100
BGBHS IMPROVED	430				-100
BGBHS IMPROVED	430				-100
BGBHS IMPROVED	430				-100
BGBHS IMPROVED	430				-100

Table 12: Instance 4.7

9.8 Instance 4.8

Method	Optimum	Min	Max	Avg	RPD
BGBHS	492				-100
BGBHS	492				-100
BGBHS	492				-100
BGBHS	492				-100
BGBHS	492				-100
BGBHS	492				-100
BGBHS	492				-100
BGBHS	492				-100
BGBHS	492				-100
BGBHS	492				-100
BGBHS IMPROVED	492				-100
BGBHS IMPROVED	492				-100
BGBHS IMPROVED	492				-100
BGBHS IMPROVED	492				-100
BGBHS IMPROVED	492				-100
BGBHS IMPROVED	492				-100
BGBHS IMPROVED	492				-100
BGBHS IMPROVED	492				-100
BGBHS IMPROVED	492				-100
BGBHS IMPROVED	492				-100

Table 13: Instance 4.8

9.9 Instance 4.9

Method	Optimum	Min	Max	Avg	RPD
BGBHS	641				-100
BGBHS	641				-100
BGBHS	641				-100
BGBHS	641				-100
BGBHS	641				-100
BGBHS	641				-100
BGBHS	641				-100
BGBHS	641				-100
BGBHS	641				-100
BGBHS	641				-100
BGBHS IMPROVED	641				-100
BGBHS IMPROVED	641				-100
BGBHS IMPROVED	641				-100
BGBHS IMPROVED	641				-100
BGBHS IMPROVED	641				-100
BGBHS IMPROVED	641				-100
BGBHS IMPROVED	641				-100
BGBHS IMPROVED	641				-100
BGBHS IMPROVED	641				-100
BGBHS IMPROVED	641				-100

Table 14: Instance 4.9

9.10 Instance 4.10

Table 15: Instance 4.10

9.11 Instance 5.1

Table 16: Instance 5.1

9.12 Instance 5.2

Table 17: Instance 5.2

9.13 Instance 5.3

Table 18: Instance 5.3

9.14 Instance 5.4

Table 19: Instance 5.4

9.15 Instance 5.5

Table 20: Instance 5.5

9.16 Instance 5.6

Table 21: Instance 5.6

9.17 Instance 5.7

Table 22: Instance 5.7

9.18 Instance 5.8

Table 23: Instance 5.8

9.19 Instance 5.9

Table 24: Instance 5.9

9.20 Instance 5.10

Table 25: Instance 5.10

9.21 Instance 6.1

Table 26: Instance 6.1

9.22 Instance 6.2

Table 27: Instance 6.2

9.23 Instance 6.3

Table 28: Instance 6.3

9.24 Instance 6.4

Table 29: Instance 6.4

9.25 Instance 6.5

Table 30: Instance 6.5

9.26 Instance A.1

Table 31: Instance A.1

9.27 Instance A.2

Table 32: Instance A.2

9.28 Instance A.3

Table 33: Instance A.3

9.29 Instance A.4

Table 34: Instance A.3

9.30 Instance A.5

Table 35: Instance A.5

9.31 Instance B.1

Table 36: Instance B.1

9.32 Instance B.2

Table 37: Instance B.2

9.33 Instance B.3

Table 38: Instance B.3

9.34 Instance B.4

Table 39: Instance B.4

9.35 Instance B.5

Table 40: Instance B.5

9.36 Instance C.1

Table 41: Instance C.1

9.37 Instance C.2

Table 42: Instance C.2

9.38 Instance C.3

Table 43: Instance C.3

9.39 Instance C.4

Table 44: Instance C.4

9.40 Instance C.5

Table 45: Instance C.5

9.41 Instance D.1

Table 46: Instance D.1

9.42 Instance D.2

Table 47: Instance D.2

9.43 Instance D.3

Table 48: Instance D.3

9.44 Instance D.4

Table 49: Instance D.4

9.45 Instance D.5

Table 50: Instance D.5

References

1. E. Talbi, *Metaheuristics - From Design to Implementation*. Wiley, 2009.
2. J. Torres-Jiménez and J. Pavón, "Applications of metaheuristics in real-life problems," *Progress in AI*, vol. 2, no. 4, pp. 175–176, 2014.
3. M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, "A racing algorithm for configuring metaheuristics," in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002* (W. B. Langdon, E. Cantú-Paz, K. E. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. K. Burke, and N. Jonoska, eds.), pp. 11–18, Morgan Kaufmann, 2002.
4. J. E. Beasley, "OR-library: distributing test problems by electronic mail," *Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1069–1072, 1990.
5. M. Birattari, L. Paquete, T. Stützle, and K. Varrentrapp, "Classification of Metaheuristics and Design of Experiments for the Analysis of Components."
6. Z. Beheshti and S. M. Hj. Shamsuddin, "Capso: Centripetal accelerated particle swarm optimization," *Inf. Sci.*, vol. 258, pp. 54–79, Feb. 2014.
7. Á. G. Rubio, B. Crawford, R. Soto, A. Jaramillo, S. M. Villablanca, J. Salas, and E. Olguín, *An Binary Black Hole Algorithm to Solve Set Covering Problem*, pp. 873–883. Cham: Springer International Publishing, 2016.
8. O. Durán, N. Rodriguez, and L. A. Consalter, "Collaborative particle swarm optimization with a data mining technique for manufacturing cell design," *Expert Syst. Appl.*, vol. 37, pp. 1563–1567, Mar. 2010.
9. B. Crawford, R. Soto, W. Palma, F. Johnson, F. Paredes, and E. Olguín, "A 2-level approach for the set covering problem: Parameter tuning of artificial bee colony algorithm by using genetic algorithm," in *Advances in Swarm Intelligence - 5th International Conference, ICSI 2014, Hefei, China, October 17-20, 2014, Proceedings, Part I* (Y. Tan, Y. Shi, and C. A. C. Coello, eds.), vol. 8794 of *Lecture Notes in Computer Science*, pp. 189–196, Springer, 2014.
10. R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia, "Local search metaheuristics for the critical node problem," *Networks*, vol. 67, no. 3, pp. 209–221, 2016.
11. R. Soto, B. Crawford, C. Galleguillos, E. Monfroy, and F. Paredes, "A hybrid ac3-tabu search algorithm for solving sudoku puzzles," *Expert Syst. Appl.*, vol. 40, no. 15, pp. 5817–5821, 2013.
12. K. Al-Ajmi and M. El-Abd, "Implementing a population-based harmony search algorithm on graphic processing units," in *IEEE 27th Canadian Conference on Electrical and Computer Engineering, CCECE 2014, Toronto, ON, Canada, May 4-7, 2014*, pp. 1–5, IEEE, 2014.
13. S. Aupetit, N. Monmarché, and M. Slimane, *Hidden Markov Models Training Using Population-based Metaheuristics*, pp. 415–438. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.
14. P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. V. Oudheusden, "A guided local search metaheuristic for the team orienteering problem," *European Journal of Operational Research*, vol. 196, no. 1, pp. 118–127, 2009.
15. B. Sarasola, K. F. Doerner, V. Schmid, and E. Alba, "Variable neighborhood search for the stochastic and dynamic vehicle routing problem," *Annals OR*, vol. 236, no. 2, pp. 425–461, 2016.
16. R. M. Karp, "Reducibility among combinatorial problems," in *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art* (M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, eds.), pp. 219–241, Springer, 2010.
17. A. I. Ali and H. Thiagarajan, "A network relaxation based enumeration algorithm for set partitioning," *European Journal of Operational Research*, vol. 38, no. 1, pp. 76–85, 1989.
18. J. J. Bartholdi, "A guaranteed-accuracy round-off algorithm for cyclic scheduling and set covering," *Operations Research*, vol. 29, no. 3, pp. 501–510, 1981.
19. W. Walker, "Using the set-covering problem to assign fire companies to fire houses," *Operations Research*, vol. 22, pp. 275–277, 1974.
20. F. J. Vasko and G. R. Wilson, "Using a facility location algorithm to solve large set covering problems," *Operations Research Letters*, vol. 3, no. 2, pp. 85–90, 1984.
21. F. J. Vasko, F. E. Wolf, and K. L. Stott, "Optimal selection of ingot sizes via set covering," *Operations Research*, vol. 35, pp. 346–353, June 1987.
22. F. J. Vasko, F. E. Wolf, and K. L. Stott, "A set covering approach to metallurgical grade assignment," *European Journal of Operational Research*, vol. 38, no. 1, pp. 27–34, 1989.
23. F. J. Vasko, F. E. Wolf, K. L. Stott, and J. W. Scheirer, "Selecting optimal ingot sizes for bethlehem steel," *Interfaces*, vol. 19, no. 1, pp. 68–84, 1989.
24. M. L. Balinski and R. E. Quandt, "On an integer program for a delivery problem," *Operations Research*, vol. 12, no. 2, pp. 300–304, 1964.

25. B. A. Foster and D. M. Ryan, “An integer programming approach to the vehicle scheduling problem,” *Operations Research*, vol. 27, pp. 367–384, 1976.
26. M. L. Fisher and M. B. Rosenwein, “An interactive optimization system for bulk-cargo ship scheduling,” *Naval Research Logistics*, vol. 36, no. 1, pp. 27–42, 1989.
27. M. Bellmore, H. J. Geenberg, and J. J. Jarvis, “Multi-commodity disconnecting sets,” *Management Science*, vol. 16, no. 6, pp. B427–B433, 1970.
28. M. Bellmore and H. D. Ratliff, “Optimal defense of multi-commodity networks,” *Management Science*, vol. 18, no. 4-part-i, pp. B174–B185, 1971.
29. B. A. Freeman and J. V. Jucker, “The line balancing problem,” *Journal of Industrial Engineering*, vol. 18, pp. 361–364, 1967.
30. M. E. Salveson, “The assembly line balancing problem,” *Journal of Industrial Engineering*, vol. 6, pp. 18–25, 1955.
31. C. C. Ribeiro, M. Minoux, and M. C. Penna, “An optimal column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment,” *European Journal of Operational Research*, vol. 41, no. 2, pp. 232–239, 1989.
32. S. Ceria, P. Nobili, and A. Sassano, “A lagrangian-based heuristic for large-scale set covering problems,” *Mathematical Programming*, vol. 81, no. 2, pp. 215–228, 1998.
33. M. A. Breuer, “Simplification of the covering problem with application to boolean expressions,” *Journal of the Association for Computing Machinery*, vol. 17, pp. 166–181, Jan. 1970.
34. N. Christofides, “Zero-one programming using non-binary tree-search,” *Computer Journal*, vol. 14, no. 4, pp. 418–421, 1971.
35. R. H. Day, “Letter to the editor—on optimal extracting from a multiple file data storage system: An application of integer programming,” *Operations Research*, vol. 13, no. 3, pp. 482–494, 1965.
36. R. S. Garfinkel and G. L. Nemhauser, “Optimal political districting by implicit enumeration techniques,” *Management Science*, vol. 16, no. 8, pp. B495–B508, 1970.
37. E. Housos and T. Elmroth, “Automatic optimization of subproblems in scheduling airline crews,” *Interfaces*, vol. 27, no. 5, pp. 68–77, 1997.
38. Z. W. Geem, J. Kim, and G. V. Loganathan, “A new heuristic optimization algorithm: Harmony search,” *Simulation*, vol. 76, no. 2, pp. 60–68, 2001.
39. Z. W. Geem, *Music-Inspired Harmony Search Algorithm: Theory and Applications*. Springer Publishing Company, Incorporated, 1st ed., 2009.
40. D. Zou, L. Gao, S. Li, and J. Wu, “Solving 0-1 knapsack problem by a novel global harmony search algorithm,” *Appl. Soft Comput.*, vol. 11, no. 2, pp. 1556–1564, 2011.
41. M. G. H. Omran and M. Mahdavi, “Global-best harmony search,” *Applied Mathematics and Computation*, vol. 198, no. 2, pp. 643–656, 2008.
42. W. Xiang, M. An, Y. Li, R. He, and J. Zhang, “An improved global-best harmony search algorithm for faster optimization,” *Expert Syst. Appl.*, vol. 41, no. 13, pp. 5788–5803, 2014.
43. M. Srinivas and L. Patnaik, “Adaptive probabilities of crossover and mutation in genetic algorithms,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 24, pp. 656–667, Apr 1994.