A Study of the Automatic Design of Heuristics for Binary Quadratic Programming

Marcelo de Souza
Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

Universidade do Estado de Santa Catarina
Rua Dr. Getúlio Vargas, 2822 – 89.140-000 – Ibirama – SC – Brasil

marcelo.desouza@udesc.br

Marcus Ritt
Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

mrpritt@inf.ufrgs.br

ABSTRACT

Researchers in the field of combinatorial optimization spend a large amount of time testing different algorithms or parameter values to solve a problem. Automatic algorithm configuration techniques perform the task of determining the best algorithm for a specific problem, and also its parameter values. In this paper we study the application of automatic methods for solving the unconstrained binary quadratic programming problem and compare it to a manual tuning process. We propose a grammar that can generate eight heuristic algorithms based on local search, tabu search, and constructive approaches, which includes the parameters of these methods. We then use an iterated F-race to find the best instantiation of the proposed grammar. For the manual tuning, we run all eight heuristics with different values of its parameters to find the best performing methods. We observe that both automatic and manual approaches can find the best algorithm candidate. However, the process is easier in the automatic way, since it requires less time and effort.

KEYWORDS. Automatic algorithm configuration. Unconstrained binary quadratic programming.

RESUMO

Pesquisadores da área de otimização combinatória dedicam grande parte do tempo testando diferentes algoritmos ou valores de parâmetros para resolver um problema. Técnicas de configuração automática de algoritmos se encarregam de determinar o melhor algoritmo para um problema específico, bem como os valores de seus parâmetros. Este trabalho apresenta um estudo sobre a aplicação de métodos automáticos para resolver o problema da programação quadrática binária irrestrita e os compara com um processo de configuração manual. Para isso, é proposta uma gramática que pode gerar oito heurísticas baseadas em busca local, busca tabu e abordagens construtivas, incluindo seus parâmetros. Este trabalho utiliza o método iterated F-race para encontrar a melhor instância da gramática proposta. Para a configuração manual, foram executados todas as oito heurísticas com diferentes valores para seus parâmetros, a fim de encontrar o método de melhor desempenho. Foi observado que ambas as abordagens, manual e automática, podem encontrar o melhor algoritmo. No entanto, o processo automático é facilitado, uma vez que requer menor tempo e esforço.

PALAVRAS CHAVE. Configuração automática de algoritmos. Programação quadrática binária irrestrita.

1. Introduction

The process of building heuristics to solve combinatorial optimization problems is complex. There are a variety of heuristic approaches, each one with its own parameters. During the design of a heuristic algorithm, the researcher has to test different approaches and find the best set of components for a specific problem (in a process called algorithm configuration), and finally find a parameter setting, that ideally should be robust, and not overly specific to the test cases. Often it is not clear from the problem at hand which heuristic strategy is the most promising, and from the no-free-lunch theorems of Wolpert and Macready (1997) we know that there's no single strategy that is always best.

Thus, researchers in this area spend a considerable part of their time testing several methods, discarding inefficient strategies, and configuring and tuning the promising ones. Furthermore, important decisions, such as which meta-heuristic strategy to apply, are often taken ad hoc, based on limited preliminary tests.

Automatic algorithm configuration is a possible way out of this situation. In this approach the researcher defines the search space of possible heuristic strategies and their feasible configurations, and delegates the task of finding a good algorithm instantiation to automatic approaches. However, these methods are not simple, because the search space of possible algorithms and parameters is large. In fact, finding a good instantiation of an heuristic algorithm in an automatic fashion is a new optimization problem and specific techniques are required.

There are many methods proposed in the literature for the automatic configuration of algorithms. Sabar et al. (2013) propose a generic hyper-heuristic with an adaptive memory for evolving online search heuristic, and test it on exam timetabling and the capacitated vehicle routing problem. Mascia et al. (2014) propose an improved mapping from a parameter space to derivations in a grammar, used for offline configuration via racing and test the method on the one-dimensional bin packing problem and permutation flowshop scheduling. Bezerra et al. (2016) propose a template for multi-objective evolutionary algorithms and evolve good heuristics via racing. A very successful method for parameter tuning is ParamILS, an iterated local search in parameter space (Hutter et al., 2007; Hutter et al., 2009). Other methods for parameter tuning are also proposed, like the application of genetic algorithms (Grefenstette, 1986; Ansótegui et al., 2009) and racing methods (Birattari et al., 2002; Balaprakash et al., 2007). Hoos (2012) surveys racing, ParamILS, and sequential methods for parameter tuning.

There are various applications of these automatic techniques in real problems. Burke et al. (2012) evolve heuristics for one-dimensional bin packing using a genetic algorithm. Drake et al. (2013) generate components for a VNS for the vehicle routing problem. Mascia et al. (2013) apply racing to generate iterated local search algorithms for permutation flowshop scheduling with weighted tardiness. Messelis and De Causmaecker (2014) use empirical hardness models to automatically select algorithms for the multi-mode resource constrained project scheduling problem. Nguyen et al. (2015) automatically design dispatching rules for job shop scheduling using iterated local search. Chiarandini et al. (2006) apply racing to configure heuristic for the university course timetabling problem. KhudaBukhsh et al. (2009) and KhudaBukhsh et al. (2016) automatically build SAT solvers from components.

In this paper we are interested in how an automatic configuration algorithm compares to a manual tuning. To this end we have chosen unconstrained binary quadratic programming (UBQP), a well-studied problem in combinatorial optimization with numerous applications for a case study. We have implemented several heuristic solution strategies for UBQP. They include simple local searches, a randomized non-monotone local search, and constructive heuristics. We have applied a typical manual calibration process, by running a set of experiments with each heuristic strategy and testing a set of previously chosen parameter settings. We also have defined a grammar, that describes the complete set of algorithm configurations and have run an automatic algorithm configuration tool. Finally, we compare both strategies, and present some conclusions with respect to automatic

configuration.

This paper is organized as follows. Section 2 presents the binary quadratic programming problem. Section 3 presents the heuristic methods implemented in this paper. In Section 4 automatic algorithm configuration process is explained in detail and discussed. Section 5 details the experiments and results. Finally, concluding remarks and future work are presented in Section 6.

2. Unconstrained Binary Quadratic Programming

Unconstrained binary quadratic programming (UBQP) is a widely used optimization problem and is defined as

$$\begin{aligned} \textbf{minimize} \quad & x^t Q x, \\ \textbf{subject to} \quad & x \in \{0,1\}^n, \end{aligned}$$

where Q is a square symmetric matrix of coefficients and x is a binary vector of unknowns. UBQP is NP-hard, even if Q is positive semi-definite (Garey et al., 1976), and even checking if an unconstrained binary quadratic program has a unique solution, or solving an UBQP which has a unique solution is NP-hard (Pardalos and Jha, 1992). The problem has multiple applications, for example in machine scheduling (Alidaae et al., 1994), in solving satisfiability problems (Hansen and Jaumard, 1990), and in problems on graphs (Kochenberger et al., 2005; Pardalos and Xue, 1994). If an optimization problem can be modeled with UBQP, it can be solved by known techniques for UBQP, without the need of problem-specific components. Kochenberger et al. (2004) show how more than twenty broad problem classes can be reduced to UBQP.

The main approaches for solving UBQP problems are presented in Kochenberger et al. (2014). They discuss several exact methods, mainly based on branch-and-bound algorithms. However, these methods are restricted to instances with a small number of variables, given the complexity of the problem. Heuristic methods can find high quality solutions to medium and large size instances. Recent studies propose several meta-heuristics like local search, tabu search, simulated annealing, and greedy heuristics. For more details about the UBQP problem, its applications, methods and theoretical results, we refer the reader to the survey of Kochenberger et al. (2014).

3. Heuristic methods for the UBQP

There are a lot of heuristic methods proposed in the literature to solve optimization problems. Since we focus on the evaluation of automatic configuration approaches, we adopt simple trajectory and construction-based heuristics, and do not intend to reach state-of-the-art results.

The first method is a simple local search with first and best improvement strategies, as well as a randomized version of a local search, which permits moves that lead to a worse solution, with a small probability. The second approach is a tabu search and a randomized tabu search. The third one is a greedy randomized adaptive heuristic, which construct solutions iteratively. By applying a local search at each solution, we obtain a GRASP heuristic. These three approaches give us eight different algorithms, which are detailed below.

3.1. Local Search

A local search begins at an initial solution and, iteratively, modifies it to improve the solution quality. This modification is performed by generating the solution neighborhood $N(s)$ and selecting one of the neighbors that improves the solution. Algorithm 1 presents a simple local search, where $B(s)$ (line 3) is the subset of $N(s)$ with neighbors better than s. The best solution found is stored in s^* (line 6).

Two main strategies are commonly applied for the selection of neighboring solution s'. The *first improvement* strategy selects the first neighbor that improves the solution quality, while the *best improvement* strategy selects the neighbor that present the best solution quality. These

Algorithm 1: Local search algorithm

Input: An initial solution s.
Output: The best solution found.

```
1 s* ← s
2 while B(s) is not empty do
3     select s' ∈ B(s)
4     s ← s'
5     if s' is better than s* then
6         s* ← s'
7 return s*
```

strategies do not select any neighbor if there is none that can improve the current solution and therefore terminate in local optima.

To overcome this problem, a randomized strategy can be applied. Algorithm 2 shows the randomized local search. Assuming $N(s)$ the set of neighbors of solution s, and $B(s)$ the set of better neighbors of solution s, the randomized strategy selects a random solution from $N(s)$ with probability p, and with probability $1 - p$ selects a random solution from $B(s)$. This method can reach better solutions, since it can overcome local optima (Hoos and Stützle, 2004).

Algorithm 2: Randomized local search algorithm

Input: An initial solution s and a probability p.
Output: The best solution found.

```
1 s* ← s
2 repeat
3     select s' ∈ N(s) or s' ∈ B(s) according to probability p
4     s ← s'
5     if s' is better than s* then
6         s* ← s'
7 until stopping criterion holds
8 return s*
```

3.2. Tabu Search

The tabu search uses an adaptive memory to guide a local search. The original proposal by Glover (1986) applies a best improvement local search. If there are no neighbors that can improve the quality of the current solution, the algorithm accepts worse solutions. A short-term memory maintaining tabu solutions is used to exclude candidate solutions from the neighborhood, avoiding cycling. To achieve this, the selected solution or an attribute of the solution is defined as tabu in the next t iterations. A simple tabu search is presented in Algorithm 3.

The randomized version of the tabu search replaces the best improvement strategy with a randomized local search, accepting worse solutions with a probability p. Different of the traditional tabu search, the randomized version has theoretical studies that prove its convergence (Faigle and Kern, 1992).

3.3. Constructive Approaches

There are many approaches based on the construction of solutions. A widely known approach are greedy algorithms, which construct a solution component by component. In each step a solution component is selected. The component that leads to the highest gain in the quality of

Algorithm 3: Tabu search algorithm

Input: An initial solution s.
Output: The best solution found.
Initialize memory M
$s^* \leftarrow s$
repeat
  select $s' \in N(s)$
  if *acceptable*(s',M) **then**
    $s \leftarrow s'$
  Update memory M
  if s' *is better than* s^* **then**
    $s^* \leftarrow s'$
until *stopping criterion holds*
return s^*

```
  <ALG>   ::=  <LS> | RLS(<p>) | TS(<t>) | RTS(<p>,<t>) | GRA(<α>) | <GRASP>
<GRASP>   ::=  GRA(<α>, <LS>)
   <LS>   ::=  FI | BI
    <p>   ::=  [0, 1]
    <α>   ::=  [0, 1]
    <t>   ::=  [0, 1]
```

Figure 1: Grammar for automatic algorithm configuration. Each derivation of this grammar represents a heuristic and its parameter settting.

the partial solution is selected, until a complete solution has been constructed. A relaxation of this method consists of defining a parameter α, and selecting one of the $\alpha\%$ best components at each step. This method is called an α-greedy algorithm.

To improve the solution quality, greedy or α-greedy algorithms can be embedded in a multi-start method. A number of solutions is constructed, and the best one is returned. We call this method GRA (Greedy Randomized Adaptive). By applying a local search at each constructed solution, we can improve its quality further. This method is called GRASP (Greedy Randomized Adaptive Search Procedure) and was proposed by Feo and Resende (1989).

4. Automatic Algorithm Configuration

We propose the application of automatic configuration techniques to find the best tuning of components and parameters to the UBQP problem. The algorithmic search space is defined by the grammar presented in Figure 1. LS is the local search heuristic with first improvement (FI) or best improvement (BI) strategies, and RLS is the randomized local search. TS is the tabu search method and RTS is its randomized version. GRA is the greedy randomized adaptive algorithm, and GRASP is the greedy randomized adaptive search procedure method.

Besides the main components represented in the grammar, some algorithms have specific parameters. The randomized versions of local search and tabu search have a parameter p that represents the probability of accepting a worse solution when selecting a neighboring solution. The tabu searches also have the tabu tenure, representing the number of iterations that a solution or a solution attribute is declared tabu. Finally, the greedy constructive approaches (GRA and GRASP) have a parameter α, which relaxes the greedy strategy to a semi-greedy approach. These parameters are also represented in the proposed grammar and its tuning can be performed by the automatic configuration approach together with the selection of the best method.

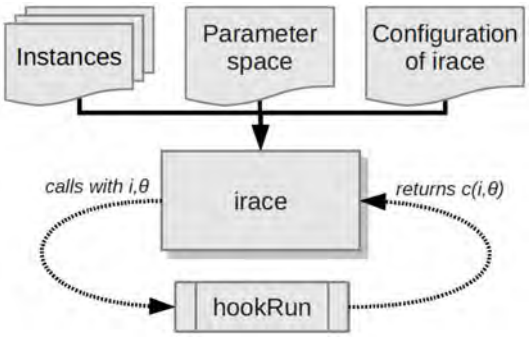

Figure 2: Operation schema of irace package. Source: López-Ibáñez et al. (2011).

```
c01     "--codon  " i (0,5)
c02     "--codon  " i (0,1) | c01 %in% c(0,5)
alpha   "--alpha  " r (0,1) | c01 %in% c(4,5)
p       "--p      " r (0,1) | c01 %in% c(1,2)
t       "--t      " r (0,1) | c01 %in% c(2,3)
```

Figure 3: Parameters configuration file `parameters.txt` used for racing with the irace package.

We have implemented a parser for the proposed grammar and a mechanism to make decisions. Due to the simplicity of the grammar, the decisions can be defined directly in the parameters to be optimized by the irace. However, we implemented a codon-based decider, a commonly used method (Mascia et al., 2014), which performs well on simple and complex grammars. This technique is based on a sequence of integer numbers, called codons. When a decision must be made, the next codon is consumed and decides which option is taken. For options $o_1, \ldots, o_k$ and codon c, option $o_{1+(c \mod k)}$ is chosen. In this way, each sequence of codons leads to a unique derivation in the grammar, which describes a heuristic algorithm. With the codon-based decision, our grammar can be easily expanded to more complex heuristic algorithms.

We used an iterated F-race (López-Ibáñez et al., 2011), implemented in the irace package for GNU R, to automatically find the best algorithm and its parameters. Figure 2 shows its operation scheme. The irace procedure receives as inputs the set of problem instances and the configuration files. We define no major settings on the irace configuration file, except the maximum number of experiments which equals to 2000. The parameter space configuration file defines the parameters for the racing algorithm to get an instantiation of the proposed grammar and tune its parameters. `hookRun` is a procedure that evaluates the grammar instantiation for a given instance. In this work, `hookRun` is a script that calls the grammar parser, recompiles the code, and runs it, returning the best found solution quality. The irace procedure uses this value in the racing method and finds a good grammar instantiation.

Figure 3 presents the parameter space configuration file used. To define a heuristic algorithm, at most two decisions are made. Thus, we use one first codon to select the main algorithm (line 1 of Figure 1) and, if necessary, a second codon to decide which specific local search strategies (line 2 and 3 of Figure 1). Besides this, we also tune the parameters of the heuristics using irace. Thus, p, α, and t are also included in the irace parameters and are used if necessary. These parameters can assume real values in the interval $[0.01, 0.99]$ and are conditioned on specific values of the first codon.

5. Experiments and Results

In this section we describe the results of computational experiments. Our main objective is to compare the results of a manual configuration and the automatic configuration. We have

Table 1: Instances used in the computational experiments.

Inst.	n	d	LB	UB	Inst.	n	d	LB	UB
bqp50-1	50	0.1	-2098	-2098	bqp250-1	250	0.1	-45607.0	-45607
bqp50-2	50	0.1	-3702	-3702	bqp250-2	250	0.1	-44810.0	-44810
bqp50-3	50	0.1	-4626	-4626	bqp250-3	250	0.1	-49037.0	-49037
bqp50-4	50	0.1	-3544	-3544	bqp250-4	250	0.1	-41274.0	-41274
bqp50-5	50	0.1	-4012	-4012	bqp250-5	250	0.1	-47961.0	-47961
bqp50-6	50	0.1	-3693	-3693	bqp250-6	250	0.1	-41014.0	-41014
bqp50-7	50	0.1	-4520	-4520	bqp250-7	250	0.1	-46757.0	-46757
bqp50-8	50	0.1	-4216	-4216	bqp250-8	250	0.1	-35726.0	-35726
bqp50-9	50	0.1	-3780	-3780	bqp250-9	250	0.1	-48916.0	-48916
bqp50-10	50	0.1	-3507	-3507	bqp250-10	250	0.1	-40442.0	-40442
bqp100-1	100	0.1	-7970	-7970	bqp500-1	500	0.1	-121588.4	-116586
bqp100-2	100	0.1	-11036	-11036	bqp500-2	500	0.1	-132216.5	-128223
bqp100-3	100	0.1	-12723	-12723	bqp500-3	500	0.1	-134214.1	-130812
bqp100-4	100	0.1	-10368	-10368	bqp500-4	500	0.1	-134781.0	-130097
bqp100-5	100	0.1	-9083	-9083	bqp500-5	500	0.1	-129572.9	-125487
bqp100-6	100	0.1	-10210	-10210	bqp500-6	500	0.1	-126429.5	-121772
bqp100-7	100	0.1	-10125	-10125	bqp500-7	500	0.1	-127136.4	-122201
bqp100-8	100	0.1	-11435	-11435	bqp500-8	500	0.1	-128574.6	-123559
bqp100-9	100	0.1	-11455	-11455	bqp500-9	500	0.1	-125821.6	-120798
bqp100-10	100	0.1	-12565	-12565	bqp500-10	500	0.1	-134352.3	-130619

implemented the heuristic algorithms in C++, and compiled them with GNU g++ version 4.2.1. All experiments have been done on a PC with a 1,7 GHz Intel Core i5 processor and 4 GB of main memory. We have chosen 40 instances proposed by Beasley (1998) with 50, 100, 250, 500 variables, with 10 instances per group. These instances are described in Table 1. The table reports for each instance the number of variables n, the density d (i.e. the fraction of non-zero elements in the matrix Q), and the best lower bound LB and best upper bound UB, as reported in the BiqMac library (Wiegele, 2007).

For the manual calibration, we have run an experiment for each main algorithm. We have executed the two local searches (first and best improvement), the randomized local search with a parameter $p \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$, the tabu search with a tabu tenure parameter $t \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$, where for value t the tabu tenure is $T = tn$, where n is the number of variables, and the randomized tabu search with every combination of $p \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$ and $t \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$. The results of these runs are shown in Tables 2 to 6. The tables report for each group of 10 instances of the same size, the average relative deviation in percent from the best upper bound.

We can see that the local searches have the worst performance, with first improvement having slightly better results than best improvement with exception of the group of instances with $n = 50$ variables. The randomized local search achieves relative deviations close or equal to 0 with $p \leq 0.25$, and $p = 0.25$ performs best. The -0.0 deviation means that a better quality solution than the best known quality values was found in some cases. The good performance for $p = 0.0$ can be explained by the large number of non-strict local minima in the search, which can be overcome by accepting moves which lead to solutions of the same quality. The tabu searches also have a good performance. The traditional tabu search performs best with $t = 1$ with relative deviations of at most 0.5, the randomized version performs best with $t = 1$ and $p = 0.0$, excluding the case $t = 0$, which corresponds to a randomized local search.

Table 2: Comparison of local searches

Alg.	Inst.	Rel.dev.(%)
BI	bqp50	5.3
BI	bqp100	3.6
BI	bqp250	2.2
BI	bqp500	2.2
FI	bqp50	6.7
FI	bqp100	3.3
FI	bqp250	2.0
FI	bqp500	1.7

Table 3: Comparison of randomized local searches

		p				
Alg.	Inst.	0	0.25	0.5	0.75	1
RLS	bqp50	0.0	0.0	0.0	0.4	21.0
RLS	bqp100	0.0	0.0	0.0	21.4	47.6
RLS	bqp250	0.0	0.0	10.7	47.5	69.0
RLS	bqp500	0.0	-0.0	28.8	59.3	77.9

For the automatic calibration we performed an iterated F-race with a budget of 2000. We have used the GNU R package irace for the experiments. The description of the parameters can be seen in Figure 3. For each parameter setting, an algorithm is generated using the grammar described above, and the corresponding heuristic is compiled on-the-fly and evaluated in 8 instances (20% of total instances). We have chosen the instances bqp50-5, bqp50-8, bqp100-5, bqp100-8, bqp250-5, bqp250-8, bqp500-5, and bqp500-8 for our experiment.

The result of the racing is shown in Figure 4. The four best configuration found by the automatic configuration procedure are all randomized local searches, which corresponds to a first codon value equal to 7. The automatic configuration finds four possible values for the parameter p as the best alternatives for it. Table 7 shows the results of the execution of the randomized local search with these values of p.

We can see that the results of the randomized local search with the parameters returned by the automatic configuration performs as well as the manually tuned variant with $p = 0.25$. The values of p found by the racing algorithm are close to 0.25. Although both the manual and automatic tuning process found equivalent configurations, the automatic approach is much more simple and performs better than the manual tuning.

With the proposed grammar and the racing parameters defined, the task of finding the best algorithm and parameters is left to irace, which uses statistical methods to explore the search space. The manual tuning is much more expensive, since it tests all algorithms and the combination

```
c01 c02 alpha p      t
7   NA  NA    0.2444 NA
7   NA  NA    0.1829 NA
7   NA  NA    0.3321 NA
7   NA  NA    0.2125 NA
```

Figure 4: Results of the iterated F-race, with the four best configurations as reported by the irace package.

Table 4: Comparison of standard tabu search strategies

		t				
Alg.	Inst.	0	0.25	0.5	0.75	1
TS	bqp50	3.7	0.0	0.4	1.1	1.3
TS	bqp100	2.6	0.1	0.8	1.5	0.1
TS	bqp250	1.8	0.5	1.2	1.2	0.0
TS	bqp500	1.7	0.9	1.2	1.0	0.1

Table 5: Comparison of randomized tabu search strategies

			t				
Alg.	Inst.	p	0	0.25	0.5	0.75	1
RTS	bqp50	0.00	4.4	0.0	0.0	0.0	1.7
RTS	bqp50	0.25	0.0	0.0	0.0	0.0	0.0
RTS	bqp50	0.50	0.0	0.0	0.0	0.5	0.0
RTS	bqp50	0.75	0.9	1.8	2.6	4.1	1.3
RTS	bqp50	1.00	21.1	22.6	22.4	22.0	22.3
RTS	bqp100	0.00	0.8	0.0	0.3	2.3	0.0
RTS	bqp100	0.25	0.0	0.0	1.1	3.8	0.0
RTS	bqp100	0.50	0.0	1.0	5.0	10.6	0.1
RTS	bqp100	0.75	19.1	20.2	21.3	23.5	19.1
RTS	bqp100	1.00	47.0	47.3	47.1	47.3	47.1
RTS	bqp250	0.00	0.5	0.3	2.4	2.0	0.0
RTS	bqp250	0.25	0.0	1.7	7.7	11.6	0.1
RTS	bqp250	0.50	3.8	13.0	19.4	26.9	8.8
RTS	bqp250	0.75	42.8	44.4	45.1	46.0	43.3
RTS	bqp250	1.00	68.7	69.1	69.3	69.0	69.3
RTS	bqp500	0.00	0.7	1.0	1.7	1.8	0.1
RTS	bqp500	0.25	-0.0	4.2	11.1	13.4	0.7
RTS	bqp500	0.50	16.0	23.1	29.3	34.7	20.2
RTS	bqp500	0.75	54.0	55.6	55.8	56.5	54.7
RTS	bqp500	1.00	77.9	78.1	77.8	77.9	77.8

of different parameter values. After all these experiments finish, the researcher has to compile all the results and analyze the best strategies. The time spent running all the necessary experiments is considerable larger than the time for the automatic approach. Moreover, the researcher has to empirically define a limited set of parameter values, while the automatic approach can test continuous intervals with a lot more precision. Finally, as the set of algorithms or the parameters space grow, manual tuning becomes impracticable, since it is not possible to test all combinations of algorithms and parameters to find the best strategy.

6. Conclusion and Future Work

In this paper we study an automatic method to design heuristic algorithms for the unconstrained binary quadratic programming problem. We use a grammar-based approach to represent the components and parameters of a set of heuristic algorithms. We then use iterated F-race to generate instantiations of the proposed grammar and finding the best candidates. Finally, we compare our automatic approach to a manual procedure of selecting the best algorithm and tuning its parameters.

Table 6: Comparison of greedy constructive strategies

		α				
Alg.	Inst.	0	0.25	0.5	0.75	1
GRA	bqp50	11.5	12.5	12.9	14.6	18.1
GRA	bqp100	2.3	1.6	1.5	2.3	6.6
GRA	bqp250	5.9	2.2	1.5	1.3	5.2
GRA	bqp500	8.4	3.7	2.7	2.4	5.2
GRASP(BI)	bqp50	11.3	11.7	12.6	14.5	18.3
GRASP(BI)	bqp100	2.3	1.4	1.5	2.2	6.6
GRASP(BI)	bqp250	5.8	2.2	1.5	1.5	5.2
GRASP(BI)	bqp500	8.6	3.7	2.7	2.4	5.2
GRASP(FI)	bqp50	10.9	11.6	13.0	14.8	18.2
GRASP(FI)	bqp100	2.3	1.4	1.5	2.2	6.6
GRASP(FI)	bqp250	5.9	2.1	1.4	1.2	5.2
GRASP(FI)	bqp500	8.5	3.7	2.7	2.3	5.2

Table 7: Comparison of randomized local searches with the parameter values automatically tuned

		p			
Alg.	Inst.	0.2444	0.1829	0.3321	0.2125
RLS	bqp50	0.0	0.0	0.0	0.0
RLS	bqp100	0.0	0.0	0.0	0.0
RLS	bqp250	0.0	0.0	0.0	0.0
RLS	bqp500	-0.0	-0.0	+0.0	-0.0

We observe that both automatic and manual strategies were able to determine the best heuristic for the proposed problem. However, the manual process was much more expensive in time and effort, since it was necessary to run experiments for all algorithms and combination of parameter values. In this work, the manual tuning performed a total of 8000 executions. This process would be unfeasible with a more complex grammar and a large number of parameters. Moreover, in the manual tuning we must define a limited set of parameter values. In the automatic tuning, the parameter values are continuous. In short, the automatic algorithm configuration can be seen as an important tool in combinatorial optimization, because it can save a lot of time of researchers in the task of explore the possible methods to solve a problem.

As future work, we will expand the proposed grammar to include new kinds of heuristic methods. Furthermore, we want to apply the automatic approach to find heuristics to other problems like Boolean satisfiability and flow shops.

References

Alidaae, B., Kochenberger, G. A., and Ahmadian, A. (1994). 0-1 quadratic programming approach for optimum solutions of two scheduling problems. *International Journal of Systems Science*, 25 (2):401–408.

Ansótegui, C., Sellmann, M., and Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In Gent, I. P., editor, *Principles and Practice of Constraint Programming*, volume 5732 of *LNCS*, p. 142–157, Heidelberg, Germany. Springer.

Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In Bartz-Beielstein, T., Blesa, M. J., Blum, C.,

Naujoks, B., Roli, A., Rudolph, G., and Sampels, M., editors, *Hybrid Metaheuristics*, volume 4771 of *LNCS*, p. 108–122.

Beasley, J. E. (1998). Heuristic algorithms for the unconstrained binary quadratic programming problem. Technical report, The Management School, Imperial College, London, England.

Bezerra, L. C. T., Lopez-Ibáñez, M., and Stützle, T. (2016). Automatic component-wise design of multi-objective evolutionary algorithms. *IEEE Trans. Evol. Comp.* forthcoming.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In et al., W. B. L., editor, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002*, p. 11–18. Morgan Kaufmann Publishers, San Francisco, CA.

Burke, E. K., Hyde, M. R., and Kendall, G. (2012). Grammatical evolution of local search heuristics. *IEEE Trans. Evol. Comp.*, 16(2):406–417.

Chiarandini, M., Birattari, M., Socha, K., and Rossi-Doria, O. (2006). An effective hybrid algorithm for university course timetabling. *J. Sched.*, 9(5):403–432.

Drake, J. H., Kililis, N., and Özcan, E. (2013). Generation of vns components with grammatical evolution for vehicle routing. In Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A. S., and Hu, B., editors, *16th European Conference on Genetic Programming*, volume 7831 of *LNCS*, p. 25–36, Vienna.

Faigle, U. and Kern, W. (1992). Some convergence results for probabilistic tabu search. *ORSA Journal on Computing*, 4(1):32–37.

Feo, T. A. and Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71.

Garey, M. R., Johnson, D. S., and Stockmeyer, L. (1976). Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Trans. Syst. Man Cyber.*, 16(1):122–128.

Hansen, P. and Jaumard, B. (1990). Algorithms for the maximum satisfiability problem. *Computing*, 44(4):279–303.

Hoos, H. H. (2012). *Autonomous search*, chapter Automated Algorithm Configuration and Parameter Tuning, p. 37–71. Springer, Berlin, Heidelberg.

Hoos, H. H. and Stützle, T. (2004). *Stochastic local search: Foundations & applications*. Elsevier.

Hutter, F., Hoos, H. H., and Stützle, T. (2007). Automatic algorithm configuration based on local search. In *Proc. of the Twenty-Second Conference on Artifical Intelligence (AAAI '07)*, p. 1152–1157.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *J. Artif. Intell. Res.*, 36:267–306.

KhudaBukhsh, A. R., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2009). Satenstein: Automatically building local search sat solvers from components. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, p. 517–524. AAAI Press, Menlo Park, CA.

KhudaBukhsh, A. R., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2016). Satenstein: Automatically building local search sat solvers from components. *Artif. Intell.*, 232:20–42.

Kochenberger, G., Hao, J.-K., Glover, F., Lewis, M., Lü, Z., Wang, H., and Wang, Y. (2014). The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization*, 28(1):58–81.

Kochenberger, G. A., Glover, F., Alidaee, B., and Rego, C. (2004). A unified modeling and solution framework for combinatorial optimization problems. *OR Spectrum*, 26:237–250.

Kochenberger, G. A., Glover, F., Alidaee, B., and Rego, C. (2005). An unconstrained quadratic binary programming approach to the vertex coloring problem. *Annals of Operations Research*, 139(1):229–241.

López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium. URL `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf`.

Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., and Stützle, T. (2013). From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness. In *7th International Conference on Learning and Intelligent Optimization*, volume 7997 of *LNCS*, p. 321–334.

Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., and Stützle, T. (2014). Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Comput. Oper. Res.*, 51:190–199.

Messelis, T. and De Causmaecker, P. (2014). An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *Eur. J. Oper. Res.*, 233:511–528.

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2015). Automatic programming via iterated local search for dynamic job shop scheduling. *IEEE Trans. Cyber.*, 45(1):1–14.

Pardalos, P. M. and Jha, S. (1992). Complexity of uniqueness and local search in quadratic 0–1 programming. *Operations Research Letters*, 11(2):119–123.

Pardalos, P. M. and Xue, J. (1994). The maximum clique problem. *Journal of global Optimization*, 4(3):301–328.

Sabar, N. R., Ayob, M., Kendall, G., and Qu, R. (2013). Grammatical evolution hyper-heuristic for combinatorial optimization problems. *IEEE Trans. Evol. Comp.*, 17(6):840–861.

Wiegele, A. (2007). Biq mac library – a collection of max-cut and quadratic 0-1 programming instances of medium size. Technical report, Institut für Mathematik, Alpen-Adria-Universität Klagenfurt.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82.