Automatic Grammar-Based Design of Heuristic Algorithms for Unconstrained Binary Quadratic Programming

Marcelo de Souza[1,2(✉)] and Marcus Ritt[2]

[1] Santa Catarina State University, Ibirama, SC, Brazil
marcelo.desouza@udesc.br
[2] Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil
marcus.ritt@inf.ufrgs.br

Abstract. Automatic methods have been applied to find good heuristic algorithms to combinatorial optimization problems. These methods aim at reducing human efforts in the trial-and-error search for promising heuristic strategies. We propose a grammar-based approach to the automatic design of heuristics and apply it to binary quadratic programming. The grammar represents the search space of algorithms and parameter values. A solution is represented as a sequence of categorical choices, which encode the decisions taken in the grammar to generate a complete algorithm. We use an iterated F-race to evolve solutions and tune parameter values. Experiments show that our approach can find algorithms which perform better than or comparable to state-of-the-art methods, and can even find new best solutions for some instances of standard benchmark sets.

Keywords: Automatic algorithm configuration
Grammatical evolution · Metaheuristics

1 Introduction

Manually developing heuristic algorithms to solve optimization problems is laborious and often biased. Engineering an effective heuristic requires to define adequate algorithmic components (e.g. neighborhoods for local searches) and the selection of a meta-heuristic, i.e. a heuristic strategy that defines how the components operate to explore the solution space (e.g. a tabu search). Since heuristic methods are difficult to analyze, there is almost no guiding theory, and the main development step consists in implementing a heuristic method and evaluating it experimentally. This is repeated in order to improve the algorithms by introducing new problem-specific components and by tuning their parameters. The experimental evaluation can take a large amount of time, which is partially spent testing ineffective components and weak heuristic strategies. At the same time good strategies may be overlooked, since heuristic design space can be very large,

A. Liefooghe and M. López-Ibáñez (Eds.): EvoCOP 2018, LNCS 10782, pp. 67–84, 2018.
https://doi.org/10.1007/978-3-319-77449-7_5

and the heuristic performance can depend on a good parameter setting. As a consequence, the design space is often not explored systematically. Additionally, designers often use shortcuts, such as short test runs, or choose a meta-heuristic only based on previous experience (e.g. a genetic algorithm, which is one of the oldest and best explored meta-heuristics). This can lead to additional biases in the design.

In order to overcome these problems, automatic algorithm engineering methods or algorithm configurators have been developed. They explore the design space and choose the best heuristic strategy combining a given set of components and finding the best parameter values. Some examples with a variety of application studies are ParamILS [1], GGA [2], SMAC [3], and irace [4].

In this paper, we focus on finding good heuristics for unconstrained Binary Quadratic Programming (BQP). Given a matrix $Q = (q_{ij}) \in \mathbb{R}^{n \times n}$ BQP asks to

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

Many problems can be reduced to binary quadratic optimization. For example, problems from machine scheduling [5], satisfiability problems [6], maximum clique problems [7], and others can be reduced to BQP. An example is the decision version of the maximum clique (MC) problem. Given an undirected graph $G = (V, E)$ and a value k, is there clique (i.e. a complete subgraph) of size k or more? With binary variables $x_v \in \{0,1\}$, $v \in V$, and weights $q_{uv} = 1$ for $\{u, v\} \in E$, and $q_{uv} = -\binom{n}{2}$ otherwise, we have a "yes"-instance of MC iff BQP has a solution of value $\binom{k}{2}$ or more. This reduction can be done in polynomial time and since MC is NP-hard shows that BQP is NP-hard, too. BQP remains hard if there is unique solution [8]. Kochenberger et al. [9] present general reduction techniques to BQP and list more than twenty problems that can be reduced to BQP.

To find better heuristics for BQP, we first extract the main problem-specific components of state-of-the art algorithms from the literature. These components include constructive heuristics, neighborhoods for local searches, perturbation strategies for iterated local searches, and solution recombination strategies, such as path relinking. We then represent the design space of meta-heuristic strategies using these components, such as iterated local searches, by a context-free grammar. A concrete strategy is represented by a complete derivation in this grammar, and thus by a sequence of decisions which rules to apply. These decision are represented by a sequence of categorical choices. In this way, the task of the automatic configuration procedure is to find good sequences of choices and good values of the parameters of the heuristic components. We then use irace as a configurator in the combined choices and parameter space and evaluate the resulting heuristic strategies.

The main contributions of this paper are: (1) an experimental study of the automatic design of heuristics for BQP; (2) the extraction of specific heuristic components for BQP from state-of-the-art algorithms; and (3) a set of competitive heuristic algorithms automatically extracted from those components.

The rest of the paper is organized as follows. Section 2 discusses related work in automatic algorithm configuration and heuristic methods for BQP. Section 3 gives details about the components we have extracted from state-of-the-art algorithms and the grammar-based approach to automatically design heuristic algorithms for BQP. Section 4 presents computational experiments and discusses the algorithms found. Finally, Sect. 5 concludes and points to future work.

2 Related Work

There are different tools for automatic algorithm configuration. ParamILS [1] applies an iterated local search in the configuration space. The method starts from the best of a number of random configurations, and then repeatedly searches a local minimum and perturbs the current solution. The local search accepts the first improvement in a neighborhood which modifies a single parameter value. The perturbation consist in a number of random moves in the neighborhood. ParamILS uses adaptive strategies to reduce the cost for comparing configurations of stochastic heuristics. GGA [2] propose a gender-based genetic algorithm for parameter tuning. Individuals represent parameter settings, and the parameters are organized in decision trees, which can represent the coupling of parameters, which is taken into account for crossover. The candidates are divided into competitive and non-competitive genders. A more recent version, GGA++ [10] guides the search using a model to estimate the empirical performance of candidates. Sequential model-based algorithm configuration (SMAC) [3] also applies a model to estimate algorithm performance. This model is used to find the most promising candidates in parameter space to evaluate next. The authors also propose a distributed version called dSMAC [11]. The irace method [4] applies iteratively Friedman-races [12] to candidates from a set of elite configurations. We have selected irace for our experiments, since it is a flexible, state-of-the-art configurator which supports categorical, ordinal, and real parameters, as well as conditional parameters. The method will be explained in more detail below.

There are many studies of the application of configurators for designing algorithms for combinatorial optimization problems. KhudaBukhsh et al. [13] present the automatic design of stochastic local search (SLS) solvers for the propositional satisfiability (SAT) problem. They apply ParamILS to find which components of SLS to use, obtaining results that outperform state-of-the-art algorithms. López-Ibáñez and Stützle [14] apply automatic algorithm configuration to design multi-objective ant colony optimization (MOACO) algorithms. They use irace to tune MOACO components and show that automatically designed algorithms can outperform MOACO algorithms found in the literature. A similar approach is found in Bezerra et al. [15], who propose the automatic design of evolutionary algorithms for multi-objective optimization problems. In both cases, the automatic approach can combine components from different algorithms of the literature, producing new state-of-the-art approaches.

Several researchers propose to represent the heuristic and parameter space by grammars [16–18]. A configurator then searches the set of decisions that leads

to the best derivation. A common strategy is to represent the choices made in the rules of the grammar by numerical values, called codons. However, several authors have pointed out that simple codon-based approaches have problems with redundancy and locality and have proposed to use rule-based tokens [19–22]. For recursive rules a limit on the number of expansions has to be defined beforehand. Mascia et al. [21,22] additionally make parameters that are not used in all derivations conditional. In this work, we follow the same approach.

BQP is NP-hard and current exact methods can only solve small instances (for more details about the exact methods see, e.g., [23]). For this reason most research focuses on heuristic methods to solve medium and large instances in the range of 2500 to about 15000 variables. Palubeckis [24] proposes an iterated tabu search. His approach iteratively perturbs the best found solution and applies a tabu search procedure. Glover et al. [25] also propose an iterated tabu search, but add a diversification strategy based on a set of elite solutions, which provides initial solutions to the search procedure. The perturbation is also based on the elite set, scoring variables according to the frequency in the elite solutions. Wang et al. [26] propose a populational heuristic that keeps an elite solution set and recombines them by two different path-relinking strategies. They also apply a tabu search to the generated solutions to improve them, updating the elite set when better solutions are found. The solution-specific components of these methods are used in our approach and will be explained in more detail in the next section.

3 Proposed Approach

Our approach is based on two main components: (1) a grammar that represents the space of heuristic strategies and their parameters, based on a given set of algorithmic components, such as constructive algorithms and neighborhoods; (2) a configurator that finds the best grammar instantiation. This section details these two components.

3.1 Grammar and the Heuristic Search Space

Algorithm configuration by grammatical evolution represents the algorithm design space by grammar. Figure 1 shows the grammar we propose to model the design space of BQP in Backus-Naur form. The grammar consists of a set of rules, through which the heuristic algorithms can be instantiated. Each rule describes a decision, i.e., a component to be chosen. The start symbol of the grammar is the non-terminal `ALG` in line 1. Starting from `ALG`, three main heuristic strategies can be chosen: heuristics based on local *search*, on solution *construction*, or on *recombination* of candidates from a population of solutions. In the following we describe all three strategies in more detail.

Search methods. The search-based heuristics (line 2) start with an initial solution and perform modifications on it, in order to explore the search space. The solutions obtained by the application of all possible modifications form a solution

```
1                 <ALG> ::= <SEARCH> | <CONSTRUCTION> | <RECOMBINATION>
2              <SEARCH> ::= LS(<IMPROVEMENT>) | NMLS(<IMPROVEMENT>) | <TS>
                            | ILS(<SEARCH>, <PERT>) | ILSE(<SEARCH>, <PERT>)
3         <IMPROVEMENT> ::= FI | FI-RR | BI | SI | SI-PARTIAL | SI-PARTIAL-RR
4                  <TS> ::= STS | RTS
5               <PERT> ::= RANDOM(<STEP>) | LEAST-LOSS(<STEP>) | DIVERSITY(<STEP>)
6               <STEP> ::= UNIFORM | GAUSSIAN | EXPONENTIAL | GAMMAM
7        <CONSTRUCTION> ::= GRA(<CONSTRUCTOR>) | GRASP(<CONSTRUCTOR>, <SEARCH>)
8         <CONSTRUCTOR> ::= ZERO | HALF
9       <RECOMBINATION> ::= RER(<IMPROVEMENT>, <SEARCH>)
```

Fig. 1. The grammar that describes the heuristic search space

neighborhood. Then, search methods apply some strategy to select the next solution from the neighborhood (line 3). Given that in the BQP the neighborhood of a solution is the set of solutions with one modified variable (a flip in a position of the vector x), we access the neighbors in the order of the variables. The first improvement (`FI`) strategy selects the first neighbor that improves the solution. We can apply a round-robin strategy (`FI-RR`), starting the exploration from the position where the previous one has finished. The best improvement (`BI`) strategy selects a neighbor that improves the solution most. The some improvement (`SI`) strategy selects a random improving neighbor. A variant, called some improvement with partial exploration (`SI-PARTIAL`) considers only the first $f\%$ of variables in the exploration for some improvement. If no improving neighbor is found, the rest of variables is explored. Alternatively, we can use a round robin strategy to start the exploration from the position where the previous one has finished (`SI-PARTIAL-RR`).

The simplest search-based method is the local search (`LS`), which iteratively applies an improving modification until no better neighbor is found. In order to avoid local optimum, a common strategy is to select a random neighbor with probability p, and an improving neighbor with probability $1 - p$. This method is called non-monotone local search (`NMLS`). Tabu search (`TS`) was first proposed by Glover [27] and consists of a local search that keeps a list of prohibited solutions (called tabu list), in order to avoid the search coming back to previous visited solutions in a short-term period. When a solution is selected, the modified variable is stored in the tabu list for some number of iterations (called the tabu tenure). During this period, this variable cannot be changed. There are different strategies to define the tabu tenure, like a constant, or a value according to the instance size. The simple tabu search (`STS`) always apply a best improvement strategy to select a neighbor. A randomized tabu search (`RTS`) applies a random move with a probability p. Commonly used stopping criteria for tabu search are a maximum number of iterations or a maximum number of iterations in stagnation. For example, Palubeckis [24] proposes the maximum number of iterations 15000 if the instance has more than 5000 variables, 12000 if it has between 3000 and 5000 variables, and 10000 for instances up to 3000 variables.

Iterated local search (`ILS`) was proposed by Lourenço et al. [28] and iteratively applies a local search, followed by a perturbation step (`PERT`). When the

search procedure ends, the current solution is at a local minimum and is perturbed in order to escape it. The iterated local search can also be combined with an elite strategy (`ILSE`). The elite set stores the best solutions found so far, which are used as initial solutions for a perturbation and a search step. We also introduce some perturbation strategies. The first one selects variables to flip at random (`RANDOM`). The second strategy ranks variables according to the loss when flipping its value. Then, the `LEAST-LOSS` approach randomly selects one of the b variables of least loss. When using an elite set, the `DIVERSITY` strategy ranks variables according to the frequency in the elite set.

Our grammar allows the combination of the iterated local search with any search and perturbation procedures. Therefore, we can instantiate state-of-the-art algorithms such as the one proposed by Palubeckis [24], which consists of an iterated local search that applies a tabu search and the least-loss perturbation procedure. We also can instantiate the diversification method of Glover et al. [25], selecting the iterated local search elite with a tabu search and a diversity perturbation procedure. In this case, the diversification-based perturbation proposed by Glover et al. scores variables using a parameter β as a weight factor for the frequency contribution, and then selects variables to be randomly assigned to 0 or 1. The probability of selection is proportional to the variable's score, using the parameter λ to define the importance of the score in this step.

The grammar also has several strategies to define the size of the perturbation (`<STEP>`), i.e., the number of variables that will be flipped. Given the instance size n, the `GAMMAM` strategy defines the perturbation size as n/g. For the other strategies, the parameters d_1 and d_2 defines the interval of possible perturbation sizes as $[d_1, n/d_2]$. The `UNIFORM` strategy selects a value of the interval according to an uniform distribution. The `GAUSSIAN` and `EXPONENTIAL` strategies apply a Gaussian and exponential distribution, respectively.

Construction methods. The construction-based methods are based on the heuristics of Merz and Freisleben [29]. They start with an empty solution and iteratively set values to its variables. The variable is chosen according to an α-greedy algorithm, which randomly selects one of the $\alpha\%$ best variables. The greedy randomized adaptive (`GRA`) algorithm repeatedly constructs m solutions and picks the best one. Another approach, proposed by Feo and Resende [30], is the greedy randomized adaptive search procedure (`GRASP`), which applies a search procedure whenever a solution is constructed. A first strategy for the construction process for BQP starts with $x = 0$, and then sets variables to one (component `ZERO` in line 8). An alternative strategy starts all variables at 0.5, and then sets the values to zero or one (`HALF`).

Recombination methods. The recombination-based method implements the idea of evolving a population of solutions. Wang et al. [26] propose the repeated elite recombination (`RER`) method, which stores an elite set with the best solutions. The elite set is initially filled with random solutions after applying the search procedure to them. Then, each pair of solutions from the elite set is recombined using path relinking. Next, a search procedure is applied to the best

Table 1. Method's parameters (n is the instance size).

Param.	Type	Method	Description	Values
t	cat	`<TS>`	Strategy for tabu tenure	$\{t_1, t_2, t_3, t_4, t_5\}$
t_v	int	`<TS>` (t_1)	Constant for tabu tenure	$[1, 50]$
t_p	int	`<TS>` (t_2)	Tabu tenure is $(t_p \times n)/100$	$[10, 80]$
t_d	int	`<TS>` (t_3 and t_4)	Tabu tenure is n/t_d	$[2, 500]$
t_c	int	`<TS>` (t_4)	Tabu tenure is $n/t_d + c \in [0, t_c]$	$[1, 100]$
s	cat	`<TS>`	Strategy for maximum stagnation	$\{s_1, s_2, s_3\}$
s_v	int	`<TS>` (s_1)	Constant for maximum stagnation	$[500, 100000]$
s_m	int	`<TS>` (s_2)	Max. stagnation is $s_m \times n$	$[1, 100]$
i	cat	`<TS>`	Strategy for maximum iterations	$\{i_1, i_2, i_3\}$
i_v	int	`<TS>` (i_1)	Constant for maximum iterations	$[1000, 50000]$
p	real	`NMLS`; `RTS`	Probability of a random move	$[0.0, 1.0]$
f	int	`SI-PARTIAL[-RR]`	Size of the partial exploration	$[5, 50]$
d_1	int	`<PERT>`	Min. perturbation size	$[1, 100]$
d_2	int	`<PERT>`	Max. perturbation size is n/d_2	$[1, 100]$
g	int	`GAMMAM`	Perturbation size is n/g	$[2, 100]$
b	int	`LEAST-LOSS`	Candidate variables for pert	$[1, 20]$
β	real	`DIVERSITY`	Frequency contribution	$[0.1, 0.9]$
λ	real	`DIVERSITY`	Selection importance factor	$[1.0, 3.0]$
r	int	`ILSE`	Elite set size for `ILSE`	$[1, 30]$
e	int	`RER`	Elite set size for `RER`	$[1, 20]$
γ	real	`RER`	Distance scale	$[0.1, 0.5]$
α	real	`<CONSTRUCTION>`	Greediness of the construction	$[0.0, 1.0]$
m	int	`<CONSTRUCTION>`	Number of repetitions	$[10, 100]$

solution found by path relinking. It replaces the worst solution in the elite set, if its quality is better. The recombination plus search step is performed repeatedly.

Given a pair of solutions, path relinking applies a local search to the first one, allowing only modifications that approximate it to the second solution. In other words, the modification of the variable i is only allowed if the new value is equal the value of variable i in the second solution. For this exploration, path relinking uses some improvement strategy. If using best improvement and no improving neighbor is found, the exploration selects the best neighbor (solution of the highest quality). When using the other strategies, if no improving neighbor is found, the exploration selects a random neighbor. Wang et al. [26] apply two different path relinking methods. The first one (PR1) uses the best improvement strategy, while the second (PR2) always selects a random neighbor. Moreover, PR1 and PR2 require that path relinking returns a solution with minimum and maximum distances from the endpoints as $d_{min} = \gamma \times H$ and $d_{max} = H - d_{min}$, where H is the Hamming distance between both solutions. If no such solution is found, the result is the starting solution s.

Parameters. Besides the algorithmic components, the grammar also defines the possible parameter values of each algorithm. For example, the constructive approaches require the definition of the number of repetitions. In this way, a complete derivation of the grammar not only gives an algorithm, but also its parameters values. Table 1 shows all parameters, their type, the related method, and possible values. Categorical parameters (like the different strategies to compute tabu tenure) have a set of limited possible values. For numerical parameters, we define the correspondent interval of possible values. Parameters t, s, and i choose the strategies for tabu tenure, maximum stagnation and maximum iterations, respectively. Strategies t_1 to t_4 are detailed in the table, and strategy t_5 is Palubeckis' rule for computing the tabu tenure. Strategy i_2 allows at most n iterations, while i_3 and s_3 allow iterations and stagnation ∞. The rest of parameters were explained above.

3.2 Automatic Design Using irace

We can see that our grammar allows to combine all the heuristic components, in order to: (1) generate the algorithms found in the literature of BQP; and (2) generate new and hybrid algorithms through the combination of these components. To perform the exploration of this search space, we use the irace tool described in [4]. irace implements an iterated racing procedure that keeps a set of elite candidates. Iteratively, it selects a candidate θ and an instance i, and then calls the target algorithm with both elements. The result metric $\varphi(\theta, i)$ is then returned to irace, which uses this value to rank different candidates. This process is repeated until irace has used the available budget of algorithm runs.

In the irace procedure, each parameter to be tuned has an associated probability distribution. Numerical and ordinal parameters have a truncated normal distribution, while categorical parameters have a discrete distribution. In each iteration, irace generates candidates according to the distribution

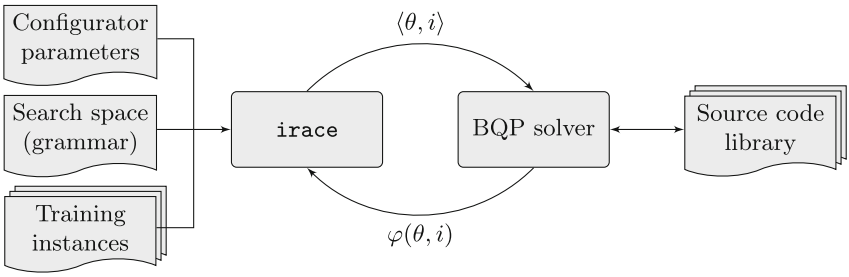

Fig. 2. General idea of the automatic algorithm configuration process

associated to their parameters, and then performs a racing to select the best candidates. Finally, the sampling distributions are updated for the next iteration, in order to bias the sampling towards the best candidates. For the selection phase, irace applies the non-parametric Friedman test to discard non-competitive configurations.

The general operation of the automatic process we use in this paper is depicted in Fig. 2. The grammar and a set of training instances are inputs to irace. We use the default values for the configurator parameters. The candidate is a complete heuristic method with the respective parameter values, and the target algorithm is the solver for the BQP. The source code library implements all heuristic components and is used by the BQP solver to run a candidate θ. We use the quality of the best found solution as result metric.

We can see that our grammar is recursive in the rule of line 2. However, unlimited depth is unhelpful and we limit the recursiveness by not allowing an iterated local search within another iterated local search. Thereby, we reduce the size of the search space defined by the grammar, but keep it possible to produce hybrid meta-heuristics, since it is based on fine-grained components and can flexibly combine them. Considering only the possible combination of components, our grammar can produce 3152 different algorithms. This number increases substantially if we consider the possible values of the 3 categorical and 15 integer parameters, and is infinite considering the 5 real parameters.

We deal with the grammar using a sequence of categorical choices representation. Each choice is an integer number, which defines the decision taken in the respective rule. For example, the rule

```
<ALG> ::= <SEARCH> | <CONSTRUCTION> | <RECOMBINATION>
```

has three possible values. A categorical choice that represents this rule will assume a value in the integer interval $[0, 2]$. Despite using integer values for each option, we follow the parametric representation presented by Mascia et al. [22] and implement these choices as categorical parameters in irace, to minimize the problem of locality. Two neighboring choice values (e.g. 0 and 1) not necessarily generate neighboring algorithms. In this example, changing the choice value

from 0 to 1 will completely change the resulting algorithm, from a search-based to a construction-based method.

A second factor that leads to locality problems is using a single sequence of choices to decide over the grammar, as adopted in [31]. In each decision, the next choice value is consumed in a round-robin strategy. This means that a single modification in some choice value can change the rule related to all next choices. To avoid this, we adopt the idea of structured representation for grammatical evolution presented in [20]. Each rule has its own sequence of choices. Thereby, a specific choice corresponds to a specific rule, independent of the values of previous choices in the grammar derivation. Moreover, since each choice is associated to the same rule, the corresponding categorical variable can assume exactly the number of choices. In addition to reducing the search space, this avoids redundancy, i.e., two choice values that lead to the same algorithm.

We also limit the maximum number of choices for each rule. This is the maximum number of times that this rule can be used to generate an algorithm. Most rules have only one choice, since they can be applied at most once. Rule `<MODIFICATION>` has two choices, because two different modification strategies can be necessary in a single algorithm (e.g. choosing the recombination method with a local search). Defining only the minimal amount of choices is possible because the grammar is non-recursive. This feature reduces the search space size, making the task of automatic configuration easier.

Finally, some choices are conditional. The choice for rule `<ALG>` has to always have a value, because this decision is taken in all possible algorithms that the grammar can generate. This is not the case for the rest of rules. For example, rule `<TS>` is applied only if a tabu search is selected in rule `<SEARCH>`. In other words, the structure of categorical choices reflects a relationship between them, defining the conditions to the need of each specific choice. We implemented these conditions, such that the configuration task needs to set values only for required choices. This feature reduces even more the search space and also avoids redundancy.

4 Experiments and Results

This section presents our experiments and discusses the obtained results. We use three different instance sets. The first is a set of 10 instances proposed by Beasley. They have 2500 variables and 10% density, i.e., approximately 10% of the elements of matrix Q have non-zero values. The second is a set of 21 instances proposed by Palubeckis [24]. They have between 3000 and 7000 variables and densities from 50% to 100%. The coefficients of these instances are uniform random integers in the interval $[-100, 100]$. The last set contains 54 instances of the MaxCut problem[1]. They have between 800 and 2000 variables and densities less than 1%. The non-zero elements assume values in $\{-1, 1\}$.

All algorithm components were implemented in C++, using the GNU GCC compiler version 5.3.1. We ran all experiments on a Linux platform running

[1] Beasley and MaxCut instances can be found in http://biqmac.uni-klu.ac.at.

on a computer with an 8-core AMD FX-8150 CPU, with 3.6 GHz and 32 GB memory. Our machine is approximately four times faster than the machine used by Palubeckis [24], so we used 25% of the time limit used by Palubeckis. This results in a time limit of 150 s for the Beasley instances and 225 s, 450 s, 900 s, 1350 s, and 2250 s for instances of Palubeckis with 3000, 4000, 5000, 6000, and 7000 variables, respectively. For the MaxCut instances, we used 66.6% (1200 s) of the time limit used by Wang et al. [26], because our machine is not more than 33% faster than their machine.

4.1 Tuning with a Single Instance Set

The density and coefficients of the Beasley and Palubeckis instances, are significantly different from those of the MaxCut instances. Therefore, we test in our first experiment an individual tuning for these two instance groups. For each of the two experiments, we used samples of the instances to tune an algorithm with a budget of 2000 runs, and then validate it using the complete set of instances. The first tuning uses the instances p3000-1, p4000-1, p5000-1, p6000-1, and p7000-1 from Palubeckis, and the second uses MaxCut instances G1, G5, G10, G15, G20, G25, G30, G35, G40, G45, and G50. We call the algorithm trained on the Palubeckis instances AAC_P, and the one trained on the MaxCut instances AAC_M.

Table 2. Average absolute gap on the different instance sets.

Inst.	ITS	D^2TS	PR1	PR2	AAC_P	AAC_M
Beasley	2.0	**0.0**	1.34	5.84	**0.0**	34.0
Palubeckis	1109.6	2082.9	457.1	690.4	**186.8**	2424.3
MaxCut	-	-	5.6	4.7	25.0	**4.4**

The results of 20 replications can be seen in Table 2. We report the average absolute gap, i.e., the difference between the quality of the found solution and the best known value, of the tuned algorithms and of the state-of-the-art algorithms ITS [24], D^2TS [25], PR1 [26], and PR2 [26]. We can see that the automatic approach can find algorithms that outperform state-of-the-art algorithms. When training using Palubeckis instances, our algorithm presents an average absolute gap of 186.8, which is much better than the average of PR1. This algorithm also works well on Beasley instances, since their structure is similar to Palubeckis instances. The AAC_P algorithm is worse than the state-of-the-art on MaxCut instances, since they have a different structure. The algorithm trained with MaxCut instances presents very good results for these instances, improving slightly over PR2. However, the performance of AAC_M is worse on the other instance sets, since it is specialized on MaxCut instances. The same behavior can be observed for algorithms PR1 and PR2 of Wang et al. [26]. PR1 performs well on Beasley and Palubeckis instances, while PR2 performs well on MaxCut instances.

Algorithm 1. AAC_R

```
while stopping criterion not satisfied do
    E ← create an elite set of size e
    while E has any novel solution do
        foreach (s, t) ∈ E × E | s ≠ t do
            V ← variables v | s[v] ≠ t[v]
            d_min ← γ × |V|
            d_max ← |V| − d_min
            d ← 0
            s* ← s
            while V ≠ ∅ do
                v ← select the best variable from V
                Flip s[v] and remove v from V
                d ← d + 1
                if (s is better than s*) ∧ (d_min ≤ d ≤ d_max) then
                    s* ← s
            s ← iteratedTabuSearch(s*) /* According to Algorithm 2 */
            if s is better than any solution of E then
                Replace the worst solution of E by s
return best solution of E
```

4.2 Tuning with a Random Instance Set

In our second experiment, we separate the training and the validation sets. To this end, we created 20 random instances for training with a similar structure than those of the Beasley and Palubeckis instances. They have 2000 to 7000 variables and densities from 10% to 100%. The coefficients were selected uniformly at random from $[-100, 100]$.

The training process using the random instance set produced a recombination-based algorithm, which internally applies an iterated tabu search. Based on the rules presented in the grammar of Fig. 1, it consists of an algorithm based on the `<RECOMBINATION>` strategy, which runs the `RER` method with the `BI` strategy for the `<IMPROVEMENT>` step, and the `ILS` as the `<SEARCH>` step. The `ILS` applies a `STS` and the `LEAST-LOSS` strategy with `GAMMAM` step for `<PERTURBATION>`. We can see that the recombination strategy is used by Wang et al. [26], while the iterated tabu search is used by Palubeckis [24]. Therefore, our automatic approach combined components of both approaches. We call this algorithm AAC_R.

The details are shown in Algorithm 1. Repeatedly, the set of elite solutions is randomly created. Each pair of solutions is selected and then recombined using path relinking. The resulting solution is improved by an iterated tabu search, and then the elite set is updated. This process is repeated while better solutions are found and added to the elite set. The path relinking method operates on the pair of solutions s and t. It searches the space of variables that, if flipped,

Algorithm 2. Iterated tabu search used in AAC_R

```
while stopping criterion not satisfied do
    p_size ← n/g
    while p_size > 0 do
        V ← create list with the b best variables to flip
        v ← random variable from V
        Flip s[v]
        p_size ← p_size − 1
    while maximum iterations and maximum stagnation not reached do
        Select variable v that leads to the best neighbor and update tabu list
        Flip s[v] and set v as tabu
return best solution found
```

approximate s to t. Iteratively, this method selects the best variable and flips it in solution s. The new solution is accepted if it is better than the best solution found so far, and according to some minimum and maximum modification steps (d_{min} and d_{max}). The iterated tabu search procedure is detailed in Algorithm 2. It applies the least loss perturbation proposed by Palubeckis [24], iteratively flipping a random variable from the b best variables to flip. This is repeated until p_{size} variables are flipped. After perturbing the solution, the search step performs a tabu search. The best found solution is stored during the perturbation and search steps, and then returned at the final of the execution. The parameter values defined by the automatic tuning are presented in Table 3.

Table 3. Parameter values tuned by the automatic approach

Parameter	t	t_d	t_c	s	s_m	i	g	b	e	γ
Value	t_4	310	25	s_2	22	i_2	10	8	16	0.2405

The results of the execution of AAC_R on the instances of Palubeckis are presented in Table 4. It presents the best and average absolute gaps obtained by AAC_R and state-of-the-art approaches. We can see that we improve the results of algorithm PR1 of Wang et al. [26], presenting an average absolute gap of 211.7. AAC_R also scales better and thus presents more uniform results, with the average absolute gap increasing less with the size of the instance.

Table 5 presents the results on MaxCut instances for algorithms AAC_R and AAC_M, as well as state-of-the-art algorithms. Column "SS" shows the results for the scatter search proposed by Marti et al. [32], and column "CC" shows the results for the CirCut method proposed by Burer et al. [33]. We can see

Table 4. Best and average absolute gaps on the instances of Palubeckis.

Inst.	ITS		D^2TS		PR1		AAC_R	
	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
p3000-1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
p3000-2	0.0	97.0	0.0	0.0	0.0	0.0	0.0	0.0
p3000-3	0.0	344.0	0.0	0.0	0.0	36.0	0.0	107.5
p3000-4	0.0	154.0	0.0	0.0	0.0	0.0	0.0	0.0
p3000-5	0.0	501.0	0.0	0.0	0.0	90.0	0.0	49.1
p4000-1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
p4000-2	0.0	1285.0	0.0	0.0	0.0	71.0	0.0	323.1
p4000-3	0.0	471.0	0.0	0.0	0.0	0.0	0.0	2.7
p4000-4	0.0	438.0	0.0	0.0	0.0	0.0	0.0	0.0
p4000-5	0.0	572.0	0.0	0.0	0.0	491.0	0.0	0.0
p5000-1	700.0	971.0	325.0	656.0	0.0	612.0	0.0	386.6
p5000-2	0.0	1068.0	0.0	12533.0	0.0	620.0	0.0	338.7
p5000-3	0.0	1266.0	0.0	12876.0	0.0	995.0	0.0	76.5
p5000-4	934.0	1952.0	0.0	1962.0	0.0	1258.0	0.0	553.6
p5000-5	0.0	835.0	0.0	239.0	0.0	51.0	0.0	36.9
p6000-1	0.0	57.0	0.0	0.0	0.0	201.0	0.0	18.4
p6000-2	88.0	1709.0	0.0	1286.0	0.0	221.0	0.0	148.4
p6000-3	2729.0	3064.0	0.0	787.0	0.0	1744.0	0.0	671.9
p7000-1	340.0	1139.0	0.0	2138.0	0.0	935.0	0.0	903.4
p7000-2	1651.0	4301.0	104.0	8712.0	0.0	1942.0	8.0	828.0
p7000-3	0.0	3078.0	0.0	2551.0	0.0	332.0	0.0	0.0
Avg.	306.8	1109.6	20.4	2082.9	**0.0**	457.1	0.4	**211.7**

that AAC_M improved the results of algorithm PR2 of Wang et al. [26]. Algorithm AAC_R have worse results for MaxCut instances compared to the other approaches. This is due to the fact that the random instances have the same structure of Palubeckis and Beasley instances, but not the density and values of the MaxCut instances. Nevertheless, our results are comparable to the state-of-the-art approaches. Moreover, algorithms AAC_M and AAC_R found new best known values for the MaxCut instances, which are presented in Table 6.

Table 5. Average absolute gap on the MaxCut instances.

Inst.	PR2	SS	CC	AAC_M	AAC_R	Inst.	PR2	SS	CC	AAC_M	AAC_R
G1	0.0	0.0	0.0	0.0	16.2	G28	7.1	11.0	36.0	8.3	10.8
G2	0.0	0.0	3.0	0.0	13.8	G29	13.1	16.0	29.0	14.8	21.6
G3	2.0	0.0	0.0	0.0	10.8	G30	7.2	9.0	27.0	6.5	14.2
G4	0.0	0.0	5.0	0.0	12.8	G31	6.5	18.0	21.0	5.3	12.2
G5	0.0	0.0	4.0	0.0	10.9	G32	5.4	12.0	20.0	8.7	34.6
G6	0.0	13.0	0.0	0.0	5.9	G33	5.9	20.0	22.0	8.7	29.8
G7	0.0	24.0	3.0	0.0	10.2	G34	5.8	20.0	16.0	8.0	31.7
G8	0.0	19.0	2.0	0.0	6.5	G35	13.2	16.0	14.0	15.2	41.5
G9	0.0	140	6.0	0.0	9.6	G36	18.3	17.0	17.0	17.0	43.5
G10	0.2	7.0	6.0	0.0	10.2	G37	21.1	25.0	23.0	19.0	44.0
G11	0.0	2.0	4.0	0.0	7.6	G38	11.6	1.0	36.0	10.3	39.5
G12	0.0	4.0	4.0	0.0	10.0	G39	15.9	14.0	12.0	11.8	65.5
G13	0.0	4.0	8.0	0.0	14.0	G40	15.7	25.0	12.0	11.2	74.1
G14	1.4	4.0	6.0	1.4	8.1	G41	15.1	18.0	6.0	11.0	74.8
G15	0.7	1.0	1.0	0.1	14.2	G42	11.8	21.0	9.0	10.2	73.1
G16	0.6	7.0	7.0	0.1	13.6	G43	0.1	4.0	4.0	0.0	10.7
G17	0.6	4.0	10.0	0.3	12.7	G44	0.1	2.0	7.0	0.0	8.6
G18	0.0	4.0	14.0	0.0	17.7	G45	0.1	12.0	2.0	0.0	6.6
G19	0.0	3.0	18.0	0.0	19.1	G46	0.2	15.0	4.0	0.0	7.5
G20	0.0	0.0	0.0	0.0	25.9	G47	0.2	8.0	1.0	0.1	9.0
G21	0.0	1.0	0.0	0.0	27.5	G48	0.0	0.0	0.0	0.0	0.0
G22	4.5	13.0	13.0	9.8	20.6	G49	0.0	0.0	0.0	0.0	0.0
G23	10.4	25.0	25.0	9.9	12.0	G50	0.0	0.0	0.0	0.0	5.0
G24	11.7	34.0	23.0	15.1	15.9	G51	1.6	2.0	11.0	1.4	15.6
G25	10.8	19.0	13.0	9.8	14.1	G52	2.6	2.0	18.0	1.1	14.9
G26	13.7	32.0	12.0	11.2	14.3	G53	2.3	4.0	8.0	2.0	14.8
G27	10.1	19.0	31.0	8.8	20.8	G54	4.2	6.0	10.0	1.9	15.7
						Avg.	4.7	10.2	10.8	**4.4**	20.3

Table 6. New best known values found.

Instance	G25	G26	G27	G28	G30	G31	G38
Previous value	13339	13326	3337	3296	3412	3306	7682
New value	13340	13327	3341	3298	3413	3310	7383

5 Conclusions

This paper presents the application of a grammar-based approach to the automatic design of heuristic algorithms for the unconstrained binary quadratic programming. The grammar describes heuristic components, including those used by state-of-the-art approaches, and the related parameters. We then use iterated F-race to explore the search space defined by the grammar and find good algorithms. Our results show the potential of automatic algorithm configuration for designing heuristic methods. It is an effective tool for the trial-and-error process of searching for a good algorithm, reducing human effort in those tasks. This potential can be seen in the AAC_R algorithm, which combines the components from different algorithms of literature into a new, competitive algorithm.

The specialized algorithms AAC_P and AAC_M present very good results, outperforming state-of-the-art approaches in the Palubeckis and MaxCut instance sets, respectively. The algorithm trained with the random instances also performs well on Palubeckis instances, and is comparable on MaxCut instances. We also note that our algorithms are not dominated by any of the algorithms from the literature. As future work, we aim at find a single algorithm that is better than the state-of-the-art approaches on both instance sets.

References

1. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intell. Res. **36**(1), 267–306 (2009)
2. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 142–157. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04244-7_14
3. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25566-3_40
4. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
5. Alidaae, B., Kochenberger, G.A., Ahmadian, A.: 0–1 quadratic programming approach for optimum solutions of two scheduling problems. Int. J. Syst. Sci. **25**(2), 401–408 (1994)
6. Hansen, P., Jaumard, B.: Algorithms for the maximum satisfiability problem. Computing **44**(4), 279–303 (1990)
7. Pardalos, P.M., Xue, J.: The maximum clique problem. J. Global Optim. **4**(3), 301–328 (1994)
8. Pardalos, P.M., Jha, S.: Complexity of uniqueness and local search in quadratic 0–1 programming. Oper. Res. Lett. **11**(2), 119–123 (1992)
9. Kochenberger, G.A., Glover, F., Alidaee, B., Rego, C.: A unified modeling and solution framework for combinatorial optimization problems. OR Spectr. **26**, 237–250 (2004)

10. Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., Tierney, K.: Model-based genetic algorithms for algorithm configuration. In: IJCAI, pp. 733–739 (2015)
11. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Parallel algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, pp. 55–70. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34413-8_5
12. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: 4th Annual Conference on Genetic and Evolutionary Computation, pp. 11–18. Morgan Kaufmann Publishers Inc. (2002)
13. KhudaBukhsh, A.R., Xu, L., Hoos, H.H., Leyton-Brown, K.: SATenstein: automatically building local search SAT solvers from components. Artif. Intell. **232**, 20–42 (2016)
14. López-Ibáñez, M., Stützle, T.: The automatic design of multiobjective ant colony optimization algorithms. IEEE Trans. Evol. Comput. **16**(6), 861–875 (2012)
15. Bezerra, L.C.T., López-Ibáñez, M., Stützle, T.: Automatic design of evolutionary algorithms for multi-objective combinatorial optimization. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 508–517. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10762-2_50
16. O'Neill, M., Ryan, C.: Grammatical evolution. IEEE Trans. Evol. Comput. **5**(4), 349–358 (2001)
17. Burke, E.K., Hyde, M.R., Kendall, G.: Grammatical evolution of local search heuristics. IEEE Trans. Evol. Comput. **16**(3), 406–417 (2012)
18. Tavares, J., Pereira, F.B.: Automatic design of ant algorithms with grammatical evolution. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 206–217. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29139-5_18
19. Rothlauf, F., Oetzel, M.: On the locality of grammatical evolution. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 320–330. Springer, Heidelberg (2006). https://doi.org/10.1007/11729976_29
20. Lourenço, N., Pereira, F.B., Costa, E.: Unveiling the properties of structured grammatical evolution. Genet. Program. Evol. Mach. **17**(3), 251–289 (2016)
21. Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T.: From grammars to parameters: automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness. In: Nicosia, G., Pardalos, P. (eds.) LION 2013. LNCS, vol. 7997, pp. 321–334. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-44973-4_36
22. Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T.: Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. Comput. Oper. Res. **51**, 190–199 (2014)
23. Kochenberger, G., Hao, J.K., Glover, F., Lewis, M., Lü, Z., Wang, H., Wang, Y.: The unconstrained binary quadratic programming problem: a survey. J. Comb. Optim. **28**(1), 58–81 (2014)
24. Palubeckis, G.: Iterated tabu search for the unconstrained binary quadratic optimization problem. Informatica **17**(2), 279–296 (2006)
25. Glover, F., Lü, Z., Hao, J.K.: Diversification-driven tabu search for unconstrained binary quadratic problems. 4OR: Q. J. Oper. Res. **8**(3), 239–253 (2010)
26. Wang, Y., Lü, Z., Glover, F., Hao, J.K.: Path relinking for unconstrained binary quadratic programming. EJOR **223**(3), 595–604 (2012)
27. Glover, F.: Tabu search. ORSA J. Comput. **1**(3), 190–206 (1989)

28. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research and Management Science, vol. 146, pp. 321–354. Springer, Heidelberg (2003). https://doi.org/10.1007/978-1-4419-1665-5_12
29. Merz, P., Freisleben, B.: Greedy and local search heuristics for unconstrained binary quadratic programming. J. Heuristics **8**(2), 197–213 (2002)
30. Feo, T.A., Resende, M.G.: Greedy randomized adaptive search procedures. J. Global Optim. **6**(2), 109–133 (1995)
31. Hyde, M.R., Burke, E.K., Kendall, G.: Automated code generation by local search. J. Oper. Res. Soc. **64**(12), 1725–1741 (2013)
32. Martí, R., Duarte, A., Laguna, M.: Advanced scatter search for the max-cut problem. INFORMS J. Comput. **21**(1), 26–38 (2009)
33. Burer, S., Monteiro, R.D., Zhang, Y.: Rank-two relaxation heuristics for max-cut and other binary quadratic programs. SIAM J. Optim. **12**(2), 503–521 (2002)