An Automatically Designed Recombination Heuristic for the Test-Assignment Problem

Marcelo de Souza
Department of Software Engineering
Santa Catarina State University – UDESC
Ibirama, Brazil
marcelo.desouza@udesc.br

Marcus Ritt
Institute of Informatics – INF
Federal University of Rio Grande do Sul – UFRGS
Porto Alegre, Brazil
marcus.ritt@inf.ufrgs.br

***Abstract*—A way of minimizing the opportunity of cheating in exams is to assign different tests to students. The likelihood of cheating then depends on the proximity of the students' desks, and the similarity of the tests. The test-assignment problem is to find an assignment of tests to desks that minimizes that total likelihood of cheating. The problem is a variant of a graph coloring problem and is NP-hard.**

We propose a new heuristic solution for this problem. Our approach differs from the usual way of designing heuristics in two ways. First, we reduce test-assignment to the more general unconstrained binary quadratic programming. Second, we search for a good heuristic using an automatic algorithm configuration tool that evolves heuristics in a space of algorithms built from known components for binary quadratic programming. The best hybrid heuristics found repeatedly recombine elements of a population of elite solutions and improve them by a tabu search. Computational tests suggest that the resulting algorithms are competitive with existing heuristics that have been designed manually.

***Index Terms*—test-assignment, binary quadratic programming, automatic algorithm configuration, metaheuristics**

I. Introduction

Given a set of desks in a classroom and a set of test variants, the test-assignment problem consists in assigning tests to desks, in order to minimize the likelihood of cheating. Each pair of desks has a known (physical) proximity, and each pair of test variants has a known similarity, and the likelihood of cheating is defined as the product of proximity and similarity. Therefore, desks that are close-by should receive less similar tests. If there are fewer students than desks in a classroom, one may additionally select a subset of free desks that remain without a test. The goal is to assign tests to desks minimizing the overall likelihood of cheating, defined as the sum of the likelihoods for each pair of desks.

The problem can be modeled as an undirected graph, whose vertices are the desks, and where pairs of desks (below a certain distance) are connected by an edge, which is weighted by the proximity of the incident desks. Each pair of test variants has an associated weight that defines their similarity. In this model it is easy to see that test-assignment generalizes the vertex coloring problem, where vertices represent the desks and each color represents a test variant. Indeed, for a given undirected graph $G = (V, E)$ we can set the proximity of all edges to 1, define k tests corresponding to k colors and set the similarity for identical tests to 1 and for different tests to 0. Then G admits a k-coloring if and only if there is an assignment of tests to desks of total likelihood 0. This implies that test-assignment is strongly NP-Hard, since vertex coloring is [1].

The test-assignment problem was introduced by Duives *et al.* [2] to improve the assignment of tests at the Engineering Faculty of the University of Bologna. The authors also have shown NP-hardness of the problem by the reduction mentioned above. To the best of our knowledge, this currently is the only published paper on the test-assignment problem. Duives *et al.* [2] formulate the problem as a non-convex binary quadratic program. Three different convex reformulations of that program are then solved with the commercial solver CPLEX: a standard reformulation, and two reformulations with stronger lower bounds obtained by solving an auxiliary semi-definite problem to partially and to optimality.

Duives *et al.* [2] further propose a tabu search that explores the space of complete colorings in a neighborhood that selects a vertex and changes its color greedily to the one that reduces the total likelihood of cheating most. The initial solution is a random feasible coloring. After changing the color of a vertex it cannot be changed again during the tabu tenure, to avoid visiting the same solution again. Different from a standard tabu search, the neighborhood is greedy and randomized: in each iteration a random vertex is chosen to change its color from a fixed percentage of the non-tabu vertices with the highest *score*, i.e. their contribution to the objective function. Unoccupied desks are taken into account by a greedy algorithm, that frees the desks of the highest score. The tabu search shows a good performance in experiments with instances up to 122 desks.

In this paper, we propose a new heuristic solution for the test-assignment problem. We reduce it to unconstrained binary quadratic programming (UBQP), and then apply the automatic algorithm configuration techniques used in our previous work [6] to search for good hybrid heuristics. The algorithmic components for the UBQP are specified by a grammar and have been extracted from the existing algorithms of literature. This approach aims at reducing human effort in the time-consuming task of searching for promising heuristics, as well as reducing the bias in manually testing algorithm components

978-1-5090-6017-7/18/$31.00 ©2018 IEEE

and their combinations. Automatic algorithm configuration for designing heuristics has been applied previously with success to several other combinatorial optimization problems, including permutation flow-shop scheduling [3], the bi-objective knapsack problem [4], boolean satisfiability problems [5], and binary quadratic programming [6].

The rest of this paper is organized as follows. Section II formalizes the test-assignment problem and gives details about the reduction to the UBQP. Section III discusses the automatic algorithm configuration task and the approach we follow to search for a good heuristic for the test-assignment. Section IV presents the algorithms produced by automatic algorithm configuration, and details the computational experiments, comparing their performance on benchmark instances to the literature. Finally, Section V presents the conclusions and outlines some future work.

II. The Test-Assignment Problem

Let D be the set of desks, T the set of test variants, and γ_{tu} the similarity between tests $t, u \in T$. We assume that only a subset $E \subseteq \{\{d, e\} \mid d, e \in D\}$ of desks are close enough, and have a defined proximity p_{de}, $\{d, e\} \in E$. We further assume that $s \leq |D|$ students will take the exam, and therefore $F = |D| - s$ desks remain free. Let $T^+ = T \cup \{0\}$ be an extended set of tests, where test "0" represents a "nonexistent" test which will be assigned to free desks. Then the test-assignment problem can be formulated as a linearly-constrained binary quadratic program as follows [2]:

$$\textbf{minimize} \quad \sum_{\{d,e\} \in E} p_{de} \sum_{t \in T} \sum_{u \in T} \gamma_{tu} x_{dt} x_{eu}, \tag{1}$$

$$\textbf{subject to} \quad \sum_{t \in T} x_{dt} = 1, \qquad d \in D, \tag{2}$$

$$\sum_{d \in D} x_{d0} = F, \tag{3}$$

$$x_{dt} \in \{0, 1\}, \qquad d \in D, t \in T. \tag{4}$$

In this model the binary variable $x_{dt} = 1$, if test $t \in T$ is assigned to desk $d \in D$, and $x_{dt} = 0$, otherwise. The model has a quadratic objective function (1), which multiplies the proximity of each pair of desks with the similarity between the assigned tests and represents the total likelihood of cheating that is to be minimized. Constraint (2) makes sure that exactly one test is assigned to each desk. Constraint (3) ensures that the "nonexistent" test is assigned to exactly F free desks.

Model (1)–(4) is clearly a linearly-constrained binary quadratic program of the form

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

by setting $Q = (q_{ij}) \in \mathbb{R}^{nm \times nm}$ with $q_{ij} = p_{de}\gamma_{tu}$, $n = |D|$, $m = |T|$, and a corresponding choice of $A \in \mathbb{R}^{(n+1) \times nm}$ and $b \in \mathbb{R}^{n+1}$.

Such a linearly-constrained binary quadratic program can be transformed to an equivalent unconstrained binary quadratic program by penalty methods (see e.g. Kochenberger *et al.* [7]). We relax $Ax = b$ and penalize the deviation from the equality in the objective function. For a large enough penalty $P \in \mathbb{R}$ the optimal solutions of the original and the transformed program (if any) will be the same. The penalty is given by

$$\begin{aligned} P(Ax - b)^t(Ax - b) &= P(x^t A^t A x - x^t A^t b - b^t A x + b^t b) \\ &= P x^t (A^t A - 2\text{diag}(A^t b))x + P b^t b. \end{aligned}$$

Given this penalty, we can define a new coefficient matrix

$$\hat{Q} = Q + P(A^t A - 2\text{diag}(A^t b)),$$

and finally rewrite the binary quadratic problem as

$$\begin{aligned} \textbf{minimize} \quad & x^t \hat{Q} x + P b^t b, \\ \textbf{subject to} \quad & x \in \{0, 1\}^{nm}. \end{aligned}$$

When optimizing, the constant term $Pb^t b$ can be omitted. To apply this transformation to the test-assignment problem, we can choose $P = \sum_{i,j|q_{ij}>0} q_{ij}$. In this way, we make sure that the best solution which violates a constraint has a larger objective value than the worst feasible solution. This reduction allows us to solve the test-assignment problem using algorithms for the UBQP.

III. Automatic Design of Heuristic Algorithms for the UBQP

In this section we explain an approach for the automatic design of heuristic algorithms for the UBQP obtained by the reduction of the test-assignment described in the previous section. To this end, we apply the automatic algorithm configuration approach of our previous work on the UBQP [6]. We have defined a grammar of meta-heuristic strategies that combines elementary components such as constructive heuristics, local searches, and solution recombination. The elementary components have been extracted from the best algorithms for the UBQP in the literature. The grammar also contains the parameters of these components, i.e. a valid derivation describes a complete heuristic algorithm.

Representing the search space of algorithmic components and parameters by grammars is a common strategy for automatic algorithm configuration. The components, their parameter values, and their combination into a heuristic are then defined by a derivation in the grammar, i.e. the decisions made in each rule. A search method can then be used to find good heuristic algorithms. A good option is to use parameter configurators for this task. Hutter *et al.* [8] propose a configurator called ParamILS, which applies an iterated local search to tune parameters. Ansótegui *et al.* [9] propose GGA, a gender-based genetic algorithm for parameter tuning. Hutter *et al.* [10] propose the use of models to guide the search for good parameter values. The same idea is used by Ansótegui *et al.* [11], applying models to guide the search of the GGA. Finally, López-Ibáñez *et al.* [12] propose irace, an iterated racing procedure for algorithm configuration. Iterated racing repeatedly samples configurations from distributions on the parameters, selects the best configurations, and adjusts the

distributions towards the best configurations. Selection is done by racing, which evaluates the configurations on a number of instances, and then applies a statistical test to eliminate the configurations that are significantly worse.

The next section details the algorithms extracted from the literature of UBQP and our approach to apply automatic algorithm configuration techniques for solving the test-assignment problem.

A. State-of-the-art algorithms for UBQP

Palubeckis [13] proposes an iterated tabu search for solving the UBQP. Given a random initial solution, it iteratively applies a tabu search as an improvement step, followed by a perturbation step to escape from local optima. The author proposes using a constant tabu tenure, and the least-loss perturbation procedure. Least-loss perturbation ranks variables according to the loss when flipping their value. Then, it randomly flips variables from the b variables of least loss. The size of the perturbation is selected uniformly at random from the interval $[d_1, n/d_2]$, where n is the size of the instance, and d_1 and d_2 are input parameters. The iterated tabu search of Palubeckis [13] presents good performance on instances up to 7000 variables.

Glover *et al.* [14] propose an iterated tabu search algorithm using an elite set as a diversification mechanism. The elite set stores the best solutions found so far, which are used as initial solutions for a perturbation and a search step. They compute the tabu tenure according to $n/t_d + c \in [0, t_c]$, where c and t_c are input parameters. The diversity-based perturbation ranks variables based on the frequency they are set to 1 in the elite solutions, and the flipping rate. The perturbation size is given by n/g, where g is an input parameter.

Wang *et al.* [15] propose a repeated elite recombination algorithm based on path relinking. The algorithm uses an elite set to store the best found solutions. Initially, it generates random solutions for the elite set and applies a tabu search procedure to improve them. Iteratively, each pair of solutions from the elite set is recombined by path relinking. The resulting solution is improved by tabu search and then replaces the worst solution of the elite set, if its quality is better. The authors propose two path relinking strategies, PR1 and PR2. PR1 uses best improvement strategy for moving to the next neighbor. If no improving neighbor is found, the exploration selects the best neighbor. On the other hand, PR2 always selects a random neighbor. Moreover, PR1 and PR2 define a minimum and maximum distance from the endpoints in the path relinking 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 path relinking returns the starting solution s. Wang *et al.* [15] present good results for the instances of Palubeckis, as well as for instances of the MaxCut problem.

B. Proposed grammar of heuristic and parameters

Given the algorithms from the state of the art of UBQP, we propose a grammar that models all their components and possible combinations. We also include several heuristic components frequently found in literature, as presented in Fig. 1. To derive a complete algorithm, we start at the rule <START> and choose some algorithmic strategy: searching a neighborhood, constructing a complete solution, or evolving a population of solutions through recombination.

The search-based heuristics iteratively explore the neighborhood of the current solution, moving from one solution to another, in order to improve its quality. A simple local search (<LS>) always selects an improving neighbor according to some strategy (<IMPROVEMENT>). As a solution is represented by a binary vector of variables, the exploration of neighbors is performed in the order of the variables. The first improvement (FI) strategy selects the first improving neighbor, and the some improvement (SI) strategy selects a random improving neighbor. The last strategy can be restricted to explore only a subset of the variables (SI-PARTIAL). A round-robin strategy (-RR) can be applied to FI or SI-PARTIAL, starting the exploration from the position where the previous one has finished. Finally, the best improvement (BI) strategy selects the neighbor that improves the solution most.

In order to escape from local optima, the non-monotone local search (NMLS) allows moving to a random neighbor with probability p, and to an improving neighbor with probability $1 - p$. The tabu search (<TS>) iteratively applies a best improvement strategy (simple tabu search – STS), but keeps a list of prohibited solutions in order to avoid the search coming back to previous visited solutions in a short-term period, called the tabu tenure (see [16]). A variant, called randomized tabu search (RTS), allows a random move with probability p. Finally, the iterated local search (ILS) component performs a local search on the current solution, followed by a perturbation to escape the local optimum. The same idea is implemented in the elite iterated local search (ILSE), which adds an elite set to provide initial solutions, as proposed by Glover *et al.* [14].

The <PERT> rule defines the available perturbation methods. The RANDOM strategy randomly selects variables and flips them. The other methods rank the variables according to the least loss (LEAST-LOSS) or frequency in the elite set solutions and flip rate (DIVERSITY), and then select variables to flip from the b best candidates. The size of the perturbation is defined by the <STEP> rule. The UNIFORM strategy has been proposed by Palubeckis [13]. The GAUSSIAN and EXPONENTIAL strategies replace the uniform distribution by a Gaussian and exponential distributions, respectively. The GAMMAM strategy defines the perturbation size as n/g, as proposed by Glover *et al.* [14].

The constructive heuristics are based on the work of Merz and Freisleben [17]. These methods start with an empty solution and iteratively assign values to the variables. The variables are randomly selected from the $\alpha\%$ best variables, and then set to zero or one. The GRA heuristic constructs m solutions and returns the best one. The GRASP does the same, but additionally applies a search procedure to each constructed solution to improve it. There are also two constructive strate-

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

Fig. 1. A grammar describing the space of heuristic algorithms.

gies (<CONSTRUCTOR>). The first is called ZERO and starts with all variables set to zero, and then sets some of them to one. The second (HALF) starts with all variables set to 0.5, and then sets each variable to zero or one. Finally, the recombination algorithm defined by the grammar is the repeated elite recombination proposed by Wang *et al.* [15]. However, the grammar allows the use of any of the improvement strategies in the recombination step, as well as any of the search procedures in the improvement step.

We can see that the grammar allows a flexible combination of its components into hybrid metaheuristics, many of which probably have never been explored before. For example, we can derive a repeated elite recombination with an iterated local search in the search step, which can use an internal non-monotone local search procedure. Moreover, we can derive the methods extracted from the literature presented above. For example, the algorithm of Palubeckis [13] can be obtained selecting the ILS algorithm with a STS improvement procedure, and a LEAST-LOSS perturbation with UNIFORM step. We can also combine the components of state-of-the-art methods with other components in order to improve their performance, as well as combine components from different methods from the literature.

Besides the algorithmic components, the automatic algorithm configurator must tune the parameters of the algorithmic components. All parameters can be found in Table I along with their type, the related method from the grammar, a short description, and the possible values. Parameter t chooses a strategy for the tabu tenure. Strategies t_1 to t_4 are explained in the table, strategy t_5 was proposed by Palubeckis [13]. It consists in selecting the tabu tenure according to the instance size. If an instance has more than 5000 variables, the tabu tenure is 15000. If it has between 3000 and 5000 variables, the tabu tenure is 12000, and 10000 for instances up to 3000 variables. The parameters s and i define strategies for the maximum iterations and maximum iterations in stagnation for tabu searches. Strategies s_3 and i_3 set these values to ∞. All other elements are input parameters of the grammar components. More details can be found in our previous work [6] and the related papers. For more details about the heuristic components, we refer to Zäpfel *et al.* [18].

C. Solution representation

As detailed in the previous sections, a solution is composed by the selected components from the grammar, and the values of the related parameters. We follow the fully parametric representation of Mascia *et al.* [19], in which the decisions made in the grammar are defined by parameters to be tuned by the configurator. For example, the <START> non-terminal in line 1 of the grammar (Fig. 1) has three options: <SEARCH>, <CONSTRUCTION>, and <RECOMBINATION>. Therefore, we define a corresponding categorical parameter with the three possible options. When defining a parameter for each non-terminal, we avoid the problems of locality and redundancy of token-based approaches (for more details about these problems we refer to Mascia *et al.* [19], Rothlauf and Oetzel [20], and Lourenço *et al.* [21]).

We also limit the number of recursions of the <SEARCH> rule, by not allowing an iterated local search being the internal search procedure of another iterated local search. This is a characteristic of the fully parametric representation, because we need a finite number of parameters. In this case, we need two parameters for the <SEARCH> rule, because when selecting one of the iterated local searches, we need to define the internal search. However, the second parameter of this rule does not have the option to select an iterated local search. Nevertheless, the proposed grammar is quite flexible and can generate a wide range of hybrid metaheuristics, because it can flexibly combine its components. In fact, the grammar can generate 3152 different algorithms. Each of them has many different possible values for its parameters, even infinite options in the case of real parameters.

Finally, we define conditions between parameters, in order to reduce the search space of components and avoid redundancy. For example, a value to the parameter that chooses a perturbation method (<PERT>) is only needed if an iterated

TABLE I
PARAMETERS OF THE DIFFERENT COMPONENTS USED THE GRAMMAR OF HEURISTIC ALGORITHMS.

Parameter	Type	Method	Description	Possible 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	$[1, 500]$
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)	Maximum 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>	Minimum perturbation size	$[1, 100]$
d_2	int	<PERT>	Maximum perturbation size is n/d_2	$[1, 100]$
g	int	GAMMAM	Perturbation size is n/g	$[2, 100]$
b	int	LEAST-LOSS	Number of candidate variables for perturbation	$[1, 20]$
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	$[10, 100]$

local search was selected (ILS or ILSE) in the rule <SEARCH>. All other components do not use perturbation steps. The same idea is applied to the input parameters, e.g., a value to parameter p is only set if a randomized heuristic was selected (NMLS or RTS).

IV. COMPUTATIONAL EXPERIMENTS

In this section we report the results of computational experiments with the proposed methods. In the first part we describe the results of the automatic algorithm configuration process, present the best found algorithm in detail, and discuss the robustness of the configuration process. In the second part we report the results of computational experiments which compare our best algorithm to existing approaches. All algorithmic components were implemented in C++ and compiled using the GNU C compiler version 5.3.1 with maximum optimization. The experiments were conducted on a PC with an 8-core AMD FX-8150 processor running at 3.6 GHz and 32 GB main memory, under Ubuntu Linux, using only one core for each execution.

In the experiments we have used the set of real-world instances proposed by Duives *et al.* [2]. It contains 36 instances based on four different classrooms ranging from 20 to 79 desks, of which between 0 and 20 were unoccupied, 50 to 250 proximity relations between neighboring desks, and two to four different tests per exam. (Duives *et al.* [2] also report results for instances with 122 desks, which were not available.)

A. Results of the Automatic Algorithm Configuration

For the automatic algorithm configuration we have randomly selected 12 instances from the complete instance set. These selected instances are marked with an asterisk in Table III. The configuration has been done using irace version 2.4.1844 with a budget of 10000 candidate runs, and a time limit of 40 seconds per run.

The tuning has been repeated two times, and the configurator found two very similar hybrid heuristic algorithms ER_1 and ER_2 with slightly different strategies and different parameters. Algorithm 1 shows the algorithmic structure of both heuristics. Repeatedly, they create an elite set with e solutions. While there are novel solutions in the elite set, a recombination by path relinking followed by a tabu search is performed for each pair of elite solutions. The path relinking process is shown in lines 5 to 19. It applies a first improvement search restricted to flipping variables that approximate the current solution s to the guiding solution t (line 5). While the solutions are different, the algorithm selects the first variable that leads to a better neighbor, flips it, and removes it from the variables to be explored. If no improving variable is found, a random variable which leads to the smallest increase of the objective function value is chosen. If the best solution found during path relinking solution is better than the incumbent solution and the number of different variables is between a minimum value d_{min} and a maximum value d_{max}, the incumbent solution is replaced by the current solution. Bounds d_{min} and d_{max} are computed based on the distance factor γ, according to lines 6 and 7. The tabu search is applied in lines 20 to 23 and uses a best improvement strategy to select non-tabu variables to flip. Finally, if the resulting solution is better than any solution of the elite population, it replaces the worst elite solution (lines 25 and 26). This algorithm could also be classified as

Algorithm 1: Hybrid heuristics ER_1 and ER_2.

```
1  while stopping criterion not satisfied do
2      E ← create an elite set of size e
3      while E has any novel solution do
4          foreach (s, t) ∈ E × E | s ≠ t do
5              V ← variables v | s[v] ≠ t[v]
6              d_min ← γ × |V|
7              d_max ← |V| − d_min
8              d ← 0
9              s* ← s
10             while V ≠ ∅ do
11                 Select the first improving variable v ∈ V in a
                    round-robin manner
12                 Flip s[v] and remove v from V
13                 d ← d + 1
14                 if s is better than s* then
15                     if d_min ≤ d ≤ d_max then
16                         s* ← s
17                     end
18                 end
19             end
20             while max. iterations/stagnation not reached do
21                 Select the best improving non-tabu variable v
22                 Update tabu list
23                 Flip s[v] and set v tabu
24             end
25             if s is better than any solution of E then
26                 Replace the worst solution of E by s
27             end
28         end
29     end
30 end
31 return best solution of E
```

a steady-state memetic algorithm with a path-relinking-based recombination operator and no mutation. It maintains diversity in the pool of elite solutions by requiring the recombined solutions to have at least $\gamma|V|$ different variables.

Table II shows the best parameter settings for ER_1 and ER_2 found in the two configuration runs. Besides the parameter values, the only difference between the two algorithms lies in the choice of the improving variable during the recombination of two solutions in line 11 of Algorithm 1: ER_1 chooses the first improving variable in a round-robin manner, i.e. it starts from the variable chosen in the previous iteration, while ER_2 chooses the first improving variable in the input variable order, always starting from the first variable. These small differences suggest that the repeated recombination followed by a tabu search is a good strategy and the configurator can reliably identify it.

B. Evaluation on the complete instance set

In this section we evaluate algorithms ER_1 and ER_2 on the complete set of instances and compare it to the tabu search proposed by Duives *et al.* [2]. The results of Duives *et al.* [2] have been obtained with a time limit of 100 s on a PC with a Pentium IV at 3.4 GHz and 2 GB main memory running Linux. For a fair comparison, we run algorithms ER_1 and ER_2 with a time limit of 40 s, to compensate for the relative performance of the two machines. The results are presented in Table III. Each instance is identified by the total number of desks, the number of unoccupied desks, and the number of different tests in the exam. For each instance we report the average absolute deviation (AD) and the relative deviation (RD) from the best known value b (defined as $v/b - 1$ for an objective function value v) for the tabu search of Duives *et al.* [2] (TS) and algorithms ER_1 and ER_2 over 20 replications with different seeds. The best relative deviations for each instance are highlighted in bold. Negative relative deviations indicate improvements over the current best known values. As mentioned above, the instances which have been used for algorithm configuration are marked with an asterisk.

TABLE II
BEST PARAMETER SETTINGS FOR HEURISTIC ALGORITHMS ER_1 AND ER_2.

Parameter	Value ER_1	Value ER_2
t	t_3	t_5
t_d	1	-
s	s_2	s_2
s_m	81	1
i	i_1	i_2
i_v	4204	-
e	20	20
γ	0.32	0.22

We can see that ER_2 leads to the best overall results, with an average relative deviation of 1.09%, followed by ER_1 with 1.33% and the tabu search with 2.90%. The newly found algorithms have a similar performance, and are significantly better than the existing tabu search, in particular on the large instances where the previous best solutions can be improved. The new best solutions found are presented in Tables IV and V. In these large instances ER_1 and ER_2 show a complementary performance, suggesting that different strategies for a small or a large number of unoccupied desks may be helpful. Both heuristics are consistently better than the tabu search on the majority of the instances, with ER_1 finding a worse solution in only 6 and ER_2 in 3 cases. In two instances TS found the best solutions.

The few instances where the tabu search has a slight advantage over ER_1 or ER_2 have 20 unoccupied desks (with the exception of the instance with 47 desks, 10 of them unoccupied, and 4 tests). This can be explained by the problem specific representation used by Duives *et al.* [2] which ignores unoccupied desks, and greedily removes exams from desks on evaluation, which simplifies the search space and the algorithm. Since we reduce the test-assignment to the UBQP, we cannot use problem-specific components. The fact that we penalize violated constraints in the objective function generates a more irregular search space, with new local optima.

TABLE III
AVERAGE ABSOLUTE AND RELATIVE GAPS ON ALL INSTANCES FOR ALGORITHMS TS, ER_1, AND ER_2.

Instance				TS		ER_1		ER_2	
Desks									
Total	Empty	Tests	Best known	AD	RD [%]	AD	RD [%]	AD	RD [%]
20	0	2	20.90	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
20	5	2 *	7.95	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
20	10	2	1.85	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
20	0	3	15.15	0.20	1.32	0.00	**0.00**	0.00	**0.00**
20	5	3	5.58	0.01	0.18	0.00	**0.00**	0.00	0.01
20	10	3 *	1.22	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
20	0	4	11.95	0.15	1.26	0.00	**0.00**	0.00	**0.00**
20	5	4	3.98	0.07	1.76	0.00	0.13	0.00	**0.00**
20	10	4 *	0.93	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
47	0	2	72.60	1.10	1.52	0.00	**0.00**	0.00	**0.00**
47	10	2	35.45	0.15	0.42	0.00	**0.00**	0.00	**0.00**
47	20	2 *	12.65	0.15	**1.19**	0.55	4.35	0.15	**1.19**
47	0	3	53.72	1.73	3.22	0.00	**0.00**	0.00	**0.00**
47	10	3 *	24.84	0.45	1.81	0.23	0.94	0.20	**0.80**
47	20	3 *	8.52	0.16	1.88	0.43	5.05	0.10	**1.17**
47	0	4	43.88	0.72	1.64	-0.05	**-0.11**	0.20	0.46
47	10	4	19.05	0.15	0.79	0.20	1.07	0.06	**0.32**
47	20	4	6.38	0.02	**0.31**	0.26	4.14	0.47	7.30
60	0	2 *	74.05	2.25	3.04	0.00	**0.00**	0.05	**0.07**
60	10	2	43.00	1.70	3.95	0.20	**0.47**	0.20	**0.47**
60	20	2 *	20.35	1.85	9.09	0.55	**2.70**	0.55	**2.70**
60	0	3 *	54.03	1.11	2.05	0.00	**0.00**	0.00	**0.00**
60	10	3	29.95	1.36	4.54	0.84	2.79	0.56	**1.88**
60	20	3	14.11	0.82	5.81	0.56	**3.97**	0.98	6.93
60	0	4	42.93	1.52	3.54	0.67	**1.56**	0.79	1.84
60	10	4	23.18	1.15	4.96	0.05	**0.22**	0.25	1.08
60	20	4	10.70	0.53	**4.95**	1.12	10.46	0.82	7.62
79	0	2 *	109.20	2.43	2.23	0.00	**0.00**	0.10	0.09
79	10	2	71.35	0.73	1.02	0.56	0.78	0.30	**0.42**
79	20	2	43.33	0.77	1.78	0.77	1.78	0.60	**1.38**
79	0	3	80.83	3.37	4.17	-0.71	**-0.88**	-0.68	-0.84
79	10	3 *	50.26	2.24	4.46	0.53	1.05	-0.26	**-0.51**
79	20	3	29.87	1.07	3.58	0.81	2.71	-0.04	**-0.13**
79	0	4	64.40	2.93	4.55	-0.19	**-0.29**	-0.04	-0.06
79	10	4	38.65	1.78	4.61	-0.22	**-0.56**	-0.07	-0.17
79	20	4 *	21.94	1.19	5.42	1.25	5.71	1.14	**5.19**
Averages			32.46	0.94	2.90	0.23	1.33	0.18	1.09

TABLE IV
NEW BEST KNOWN VALUES FOUND BY ALGORITHM ER_1.

Instance			Solution values	
Desks				
Total	Empty	Tests	Previous value	New value
47	10	3	24.84	24.64
47	0	4	43.88	43.83
47	10	4	19.05	19.03
79	0	3	80.83	80.12
79	10	3	50.26	50.01
79	0	4	64.40	64.10
79	10	4	38.65	38.00

TABLE V
NEW BEST KNOWN VALUES FOUND BY ALGORITHM ER_2.

Instance			Solution values	
Desks				
Total	Empty	Tests	Previous value	New value
47	10	4	19.05	18.93
79	0	3	80.83	80.03
79	10	3	50.26	49.71
79	20	3	29.87	29.83
79	0	4	64.40	64.33
79	10	4	38.65	38.15

For example, to go from a feasible solution to another, a constraint must be violated and, consequently, the quality of the intermediate solutions will be worse. The algorithm must handle this new search space topology.

Finally, we have evaluated the contribution of the effectiveness of the configuration tool for finding good heuristic algorithms. To this end, we generated ten random hybrid heuristics from the grammar and evaluated them on the complete instance set under the same conditions. We found that the random heuristics have an average relative deviation of $46.5\,\%$ with a standard deviation of $31.1\,\%$. This shows that

the search space of algorithms contains heuristics of strongly varying quality and that the automatic algorithm configuration is effective in finding heuristics that perform well.

V. Conclusions

In this paper we have proposed a new metaheuristic to solve the test-assignment problem. We reduce test-assignment to unconstrained binary quadratic programming, which in turn is solved by hybrid heuristics based on algorithmic components which have proven to work well in the literature. Instead of a manually, laborious and often biased search for the best heuristic methods, we apply automatic algorithm configuration to perform this task. We have extracted algorithmic components and parameters from state-of-the-art approaches and represent them in a grammar. Then, we apply the automatic configurator irace to explore this search space and find good combinations of components and parameter values. The obtained algorithms ER_1 and ER_2 perform well on the existing benchmark instances, improve the state-of-the-art method and are able to find new best solutions on the more difficult instances with a large number of desks.

This study contributes to the evidence that automatic algorithm configuration techniques can reduce the effort of researchers compared to a manual configuration, allowing them to focus on other tasks, such as the design of new algorithmic components. We also show that heuristics for the UBQP can be successfully applied to quadratic binary optimization with linear constraints. This makes the proposed grammar and the automatic methods a promising tool for solving a large class of problems that can be modeled in this way. In future work we intend to include problem-specific components in the grammar, e.g. a specialized neighborhood, in order to improve the results for instances with large number of unoccupied desks.

References

[1] M. R. Garey and D. S. Johnson, "Two-processor scheduling with start-times and deadlines," vol. 6, no. 3, pp. 416–426, Sep. 1977.

[2] J. Duives, A. Lodi, and E. Malaguti, "Test-assignment: A quadratic coloring problem," *Journal of heuristics*, vol. 19, no. 4, pp. 549–564, 2013.

[3] F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle, "From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness," in *International Conference on Learning and Intelligent Optimization*, Springer, 2013, pp. 321–334.

[4] L. C. T. Bezerra, M. López-Ibánez, and T. Stützle, "Automatic generation of multi-objective ACO algorithms for the bi-objective knapsack," in *International Conference on Swarm Intelligence*, Springer, 2012, pp. 37–48.

[5] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown, "SATenstein: Automatically building local search SAT solvers from components," *Artificial Intelligence*, vol. 232, pp. 20–42, 2016. DOI: 10.1016/j.artint.2015.11.002.

[6] M. de Souza and M. Ritt, "Automatic grammar-based design of heuristic algorithms for unconstrained binary quadratic programming," in *Evolutionary Computation in Combinatorial Optimization*, Springer, Springer International Publishing, 2018, pp. 67–84, ISBN: 978-3-319-77449-7. DOI: 10.1007/978-3-319-77449-7_5.

[7] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang, "The unconstrained binary quadratic programming problem: A survey," *Journal of Combinatorial Optimization*, vol. 28, no. 1, pp. 58–81, 2014.

[8] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An automatic algorithm configuration framework," *Journal of Artificial Intelligence Research*, vol. 36, no. 1, pp. 267–306, 2009.

[9] C. Ansótegui, M. Sellmann, and K. Tierney, "A gender-based genetic algorithm for the automatic configuration of algorithms," *Principles and Practice of Constraint Programming*, pp. 142–157, 2009.

[10] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," *Learning and Intelligent Optimization Conference*, vol. 5, pp. 507–523, 2011.

[11] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney, "Model-based genetic algorithms for algorithm configuration.," in *IJCAI*, 2015, pp. 733–739.

[12] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[13] G. Palubeckis, "Iterated tabu search for the unconstrained binary quadratic optimization problem," *Informatica*, vol. 17, no. 2, pp. 279–296, 2006.

[14] F. Glover, Z. Lü, and J.-K. Hao, "Diversification-driven tabu search for unconstrained binary quadratic problems," *4OR: A Quarterly Journal of Operations Research*, vol. 8, no. 3, pp. 239–253, 2010.

[15] Y. Wang, Z. Lü, F. Glover, and J.-K. Hao, "Path relinking for unconstrained binary quadratic programming," *EJOR*, vol. 223, no. 3, pp. 595–604, 2012.

[16] F. Glover, "Tabu search," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[17] P. Merz and B. Freisleben, "Greedy and local search heuristics for unconstrained binary quadratic programming," *Journal of Heuristics*, vol. 8, no. 2, pp. 197–213, 2002.

[18] G. Zäpfel, R. Braune, and M. Bögl, *Metaheuristic search concepts: A tutorial with applications to production and logistics*. Springer Science & Business Media, 2010.

[19] F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle, "Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools," *Computers & Operations Research*, vol. 51, pp. 190–199, 2014.

[20] F. Rothlauf and M. Oetzel, "On the locality of grammatical evolution," in *European Conference on Genetic Programming*, Springer, vol. 3905, 2006, pp. 320–330.

[21] N. Lourenço, F. B. Pereira, and E. Costa, "Unveiling the properties of structured grammatical evolution," *Genetic Programming and Evolvable Machines*, vol. 17, no. 3, pp. 251–289, 2016.