The Template Design Problem: A Perspective with Metaheuristics

David Rodríguez Rueda[1], Carlos Cotta[2], and Antonio J. Fernández-Leiva[2]

[1] Universidad Nacional Experimental del Táchira (UNET), Laboratorio de Computación de Alto Rendimiento (LCAR), San Cristóbal, Venezuela, drodri@unet.edu.ve

[2] Universidad de Málaga, ETSI Informática, Campus de Teatinos, 29071 Málaga, Spain, {ccottap,afdez}@lcc.uma.es

Abstract. This paper deals with the template design problem, a hard constrained combinatorial problem with multiple applications. This problem is here formulated as a two-level combinatorial optimization problem whose solutions are integer matrices. In the higher level, a metaheuristic tackles the design of a collection of templates containing multiple instances of a set of components to be produced; in the lower level an integer linear programming solver is used to determine the optimal number of times each template has to be pressed in order to fulfill production requirements as closely as possible. Three metaheuristics, i.e., hill climbing, tabu search, and genetic algorithms, have been considered in the higher level, and LPSolve, a simplex-based software for linear and integer programming problems, in the lower level. An empirical evaluation on three scenarios of increasing complexity has been performed, indicating the better performance of genetic algorithms. These results are comparable to those shown by sequential ILP models, and hint the possibility of hybrid approaches.

1 Introduction

Many problems in the area of manufacturing are related to reducing the waste of the raw material used in the production process. This is precisely the case of the Template Design Problem (TDP) considered in this article. The TDP arises in industrial settings in which variations of certain products must be produced, each of them requiring a particular packaging (typically with different printing patterns). The production of these packages must be accomplished minimizing the use of cardboard (or any other raw material used). Appropriate templates for printing these packages must then be designed, hence leading to the TDP – see Sect. 2 for a more detailed description of the problem.

Several methods have been proposed in the literature to attack this problem, which turns out to be extremely hard. Our proposal is based on the use of metaheuristics, which to the best of our knowledge is a novel approach for this problem. More specifically, we consider both local search methods –i.e., a steepest descent hill climbing (HC) algorithm and tabu search (TS)– and population-based techniques, i.e., a genetic algorithm (GA) – see Sect. 3. These techniques

are deployed on the TDP aiming to find adequate template designs. Such designs must be evaluated in terms of the optimal usage that can be done of them. For this purpose, an ILP model shown by Proll and Smith [8] is used and solved with an exact method, hence leading to a two-level optimization approach. As it will be shown in Sect. 4, the resulting techniques can effectively provide high quality solutions when applied to several instances taken from the literature.

2 The Template Design Problem

The template design problem was first described by Proll and Smith [8] who observed this problem arising at a local color printing firm. Roughly speaking, we can assume a certain product is manufactured with several variations (e.g., cereal flakes of different flavors), each one requiring a similar –but different– packaging. In order to produce the latter, a printing machine is used. This machine is configured with a certain template, which is subsequently pressed on sheets of raw material (e.g., cardboard). Given the large number of items required, a template comprises several slots, each of them filled with a certain variation of the product, which are printed on each pressing. In addition, there can be more than one such template. This means that the problem is twofold: (i) determine the design of each template, namely which variations are included in each slot, and (ii) determine the optimal usage of these templates. The last point requires a certain criterion to be optimized, for example minimizing the manufacturing time (i.e., minimizing the number of pressings) or minimizing the waste, given the known demands of each variation. We consider here this latter criterion, i.e., optimize the use of raw material.

In order to solve subproblem (ii) above, an ILP formulation provided by Proll and Smith is used. Let V be the number of variations to be produced; let there be T templates $T_1, \cdots, T_n$, each of them with S slots. Notice that the particular slot in which a variation is placed is irrelevant: it only matters how many instances of a certain variation are contained in a given template. Thus, let p_{ij} be the number of instances of variation i in template T_j. Now, let Q_i be the demand for variation i (deterministic and known; see [6] for an approach under uncertainty), and let us assume that we have production tolerances l_i, u_i for each variation, i.e., we can afford up to a certain underproduction $Q_i l_i$ or overproduction $Q_i u_i$ for each variation i. Then, the resulting problem is

$$\varphi = \min_{i=1\ldots V} \sum (U_i + O_i) \tag{1}$$

subject to:

$$\sum_{j=1\ldots T} p_{ij} R_j + U_i - O_i = Q_i, \quad 1 \leqslant i \leqslant V \tag{2}$$

$$\sum_{j=1\ldots T} p_{ij} R_j \geqslant (1 - l_i) Q_i, \quad 1 \leqslant i \leqslant V \tag{3}$$

$$\sum_{j=1\ldots T} p_{ij} R_j \leqslant (1 + u_i) Q_i, \quad 1 \leqslant i \leqslant V \tag{4}$$

$$R_j \geqslant 0, \quad 1 \leqslant j \leqslant T \tag{5}$$

$$U_i, O_i \geqslant 0, \quad 1 \leqslant i \leqslant V \tag{6}$$

where U_i and O_i are slack variables that respectively represent the underproduction and overproduction of variation i, and R_j denotes the number of times template j is punched (pressed). This ILP problem can be solved using an appropriate exact technique such as branch and bound or branch and cut. Of course, there still remains the problem of determining the design of the templates, used as input in this ILP formulation. This is precisely where metaheuristics come into play.

3 Solving the TDP under Deterministic Demand

The TDP exhibits a clear combinatorial structure, and can be readily transformed in an optimization task. We have approached this challenging resolution via two local search techniques (HC and TS) and a population-based technique (GA), which will be described below. To this end, let us firstly define the representation used, as well as possible neighborhood structures.

3.1 Representation and Evaluation

The matrix model provides a very natural way of representing the problem. Taking the model suggested by [10] each candidate solution is represented by a matrix $M = \{p_{ij}\}_{V \times T}$, where as mentioned before $p_{ij} \in \{1, \cdots, S\}$ indicates the number of copies of variation i in template T_j. We enforce the constraint

$$\forall j : \sum_i p_{ij} = s. \tag{7}$$

Notice the inherent symmetry of the representation: columns (i.e., templates) can be arbitrarily reordered. However, this representation does break the symmetry inside templates (i.e., it only matters how many but not which slots are used by each variation).

The initialization of candidate solutions can be done randomly (i.e., assigning a random variation to each slot) or via some ad-hoc heuristic. In the latter case, we can consider the process shown in Algorithm 1. It essentially amounts to a biased random initialization in which variation are given slots with a probability proportional to the fraction of the total demand they represent. A correction mechanism was also used to avoid that some variation was associated to no slot. This means to impose the constraint

$$\forall i \in V : \sum_{j=1}^{T} p_{ij} > 0. \tag{8}$$

Given this representation, a simple neighborhood has been considered by reassigning a single slot in a template to a different variation. Given the constraint

Algorithm 1: Pseudo code of initialization under demand (ID)

```
1  begin
2      p ← InitializeTemplate(T, V);
       // Computes the fraction of the demand for each variation.
3      pq ←ComputeProbabilities(Q);
4      for i ← 1 to T do
           // Fills the i-th template according to pq.
5          RouletteFill(p, pq, i, V);
           // Computes a heuristic number of pressings for the i-th
           // template and updates probabilities according to the
           // remaining demand.
6          R ← min_{j=1...V} {(1 − L_j)Q_j/p_ij | p_ij > 0};
7          for j ← 1 to V do
8              Q_j ← Q_j − R ∗ p_ij;
9          end for
10         pq ←ComputeProbabilities(Q);
11     end for
12 end
```

mentioned previously, this means that a variation with non-zero appearances in a certain template must be selected (i.e., find a certain $p_{ij} > 0$), its count number has to be decreased and subsequently the number of copies of another variation in the same template must be increased. For the sake of efficiency, the variation i whose number of copies is decreased is selected such that $\sum_j p_{ij} > 1$ since otherwise its total count number across all templates could drop to zero and hence no copy of it could be printed at all – recall Eq. (8). Notice that the maximum size of the neighborhood is $\Theta(TV^2)$.

In order to evaluate a certain solution, the ILP model is solved by using `lp_solve` [2], a simplex-based software for linear and integer programming problems, and the returned value (the waste) is minimized. Notice that it may be possible that no feasible solution exists for an arbitrary configuration of templates, i.e., the target production cannot be reached within the desired tolerances. If the solver indicates that this is the case, the problem is relaxed by removing the overproduction constraint and a penalty term is added.

3.2 Metaheuristic Approaches

The hill climbing (HC) approach and a tabu search [3, 4] (TS) algorithm were defined on the basis of the neighborhood structure mentioned before.

The HC algorithms follow a steepest-descent procedure: the neighborhood of the current solution is partially explored and the best solution is chosen unless it is worse than the current one; if this is the case, the current solution is a local optimum, and the process is re-started from a different point (randomly or heuristic chosen) until the computational budget allocated is exhausted. Regarding the TS algorithm, the neighborhood is handled in the same way as in

the algorithm HC, moving to the best non-tabu neighbor even if it is worse than the current solution. A move is tabu if attempts to reverse a slot reassignment $\{p_{ij}^{++}, p_{kj}^{--}\}$ previously done and stored in the tabu list; here the notation p_{ij}^{++} (resp. p_{kj}^{--}) indicates that the number of copies of variation i (resp. k different from i) in template j have been increased (resp. decreased) in one occurrence. To prevent cycling, the tabu tenure –i.e., the maximum number of iterations a tabu move stays in the list– is chosen randomly in the range $[\beta/2, 3\beta/2]$, where $\beta = V^2T$. The tabu status of a move can be overridden if the aspiration criteria is fulfilled, namely, finding a solution better than the current best solution found so far. After a number of n_ι evaluations (a parameter that we will set as a function of the total number of evaluations) with no improvement, the search is intensified, by returning to the best solution found so far.

In addition to the local search approaches mentioned before, a genetic algorithm [1] (GA) is also considered. This GA handles a population of candidate solutions that are iteratively selected, bred and replaced. To be precise selection is done by binary tournament, and replacement is done following a $(\mu, 1)$–policy, i.e., a new individual is generated and inserted in the population replacing the worst one. As to breeding, it is done by recombination and mutation as usual. The recombination operator used is a variant of uniform crossover (UX) [9]: firstly, the templates are compared to find the best matching among them (recall the symmetry of the problem, and the fact that templates can be arbitrarily reordered). Subsequently, a template-level exchange of information is done, by randomly choosing a template out of each pair of matched templates (one from each of the parents). Notice that this amounts to a column-wise exchange in matrix $M = \{p_{ij}\}$. The mutation is handled the same way as in the previously mentioned neighborhood.

A final remark must be done regarding the overall resolution approach: an incremental process is attempted, by trying first to solve the problem with a low number of templates (e.g., one template), and then using the solutions found in this scenario as initial solutions (either for local search or for inclusion in the GA population) for the resolution of the problem with a higher number of templates. To this end, the candidate solutions must be obviously completed (e.g., a solution for the problem with $T = 1$ needs to be expanded with an additional template to be a tentative solution for the problem with $T = 2$). this is done following the same initialization techniques sketched above.

4 Experimental results

Three instances of the problem, with the same demand requirements as those described in [8], has been considered for the experiments. Table 1 shows a summary of the results obtained by Proll and Smith (see Tables 3 and 7 and equations from (7)–(10) in [8].

All our algorithms have been run 20 times per problem instance. The number of evaluations for each scenario is subject to the number of variations and templates: $n_\nu = 1000 \cdot T \cdot V \cdot (V-1) \cdot \%_v$, where $\%_v$ represents the percent-

Table 1. Result for the TDP instances in [8].

	sequential ILP solutions		
Problem	No. Template	Overall % Dev.	Maximum % Dev.
Cat Food Cartons	3	−2.59	−7.27
Herbs Cartons	3	−2.91	−8.70
Magazine Inserts	4	−2.49	−10.00

	goal program adjustment of constraint programming solutions		
Problem	No. Template	Overall % Dev.	Maximum % Dev.
Cat Food Cartons	3	−1.4	−1.9
Herbs Cartons	2	−3.2	$+10/-6.7$
Magazine Inserts	3	−2.4	$+7.4/-8.3$

Table 2. Results for Hill Climbing (20 runs per instance).

Problem	Number of Templates	Templates	Pressings	Overall Desv.	T.Desv
Cat Food	2	[0,0,0,0,0,2,7]	157143		
		[1,1,1,2,2,2,0]	250000	0.80	-3.85/ 1.79
	3	[0,1,1,0,0,7,0]	7143		
		[0,0,0,0,0,5,4]	150000		
		[1,1,1,2,2,0,2]	250000	0.14	-1.10/ 0.84
Hb.Cartons	2	[0,0,0,0,0,0,0,0,0,1,1,1,1,2,1, 1,1,1,1,2,2,2,2,5,2,2,2,2,5,5]	16669		
		[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,1,3,3,3,3,3,3]	65554	2.98	-8.64/ 9.88
	3	[0,0,2,2,1,2,2,2,2,2,0,0,0,0,3,3, 3,0,3,3,1,1,1,1,2,1,1,1,1,2,2]	3333		
		[0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1, 1,2,1,1,2,2,2,2,1,6,6,2,2,1,5]	16666		
		[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,2,2,2,3,3,4,3]	63335	0.76	-4.76/ 7.41
Mz.Inserts	3	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,2,0, 3,3,0,3,3,0,0,3,0,0,0,0,0,1,0,1,0,1,2,2,1,2,3,3,0]	21666		
		[0,0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0, 1,1,1,1,1,1,1,0,1,1,2,0,2,1,0,1,0,1,1,1,0,1,1,1,3]	135000		
		[1,1,1,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,2, 0,0,1,0,0,1,1,2,1,1,0,3,0,1,3,1,3,1,1,1,3,1,2,2,0]	83333	5.83	-10.00/ 66.67
	4	[0,0,0,0,1,1,0,2,0,0,1,1,1,1,2,2,0,0,0,0,0,0,1,0,1, 0,0,0,0,0,1,1,0,2,0,0,2,0,2,2,2,2,2,2,0,1,1,1,2,2,3]	12500		
		[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,2,0, 3,3,0,3,3,0,0,3,0,1,0,0,0,1,0,1,0,1,2,2,1,2,3,3,0]	25000		
		[0,0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0, 1,1,1,1,1,1,1,0,1,1,2,0,2,1,0,1,0,1,1,1,0,1,1,1,3]	125000		
		[1,1,1,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,2, 0,0,1,0,0,1,1,2,1,1,0,3,0,1,3,1,3,1,1,1,3,1,2,2,0]	75000	2.60	-4.41/ 50.00

age of neighbors to be evaluated (note that a full exploration schema of the neighborhood is very costly) with $V = 50$ and $T = 4$, evaluating just 5% of the neighborhood means to evaluate $4.9 \cdot 10^5$ neighbors- and thus a partial exploration policy was considered). For LS algorithms we set $n_\iota = n_\nu/10$. As to the GA, we set *population size*= 100, crossover and mutation probabilities $p_X = .9$ and $p_M = 1/(VT)$.

Tables 2 and 3 show the results obtained by applying HC and TS respectively. Initially the LS algorithms were executed with $\%_v = 5\%$ and $\%_v = 1\%$ respectively for the instances *Herbs Cartons* and *Magazine Insert* but the performance was poor; as to the less complex scenario, that is to say *Cat food*, we employed a full neighborhood exploration schema but again the results were really poor. After executing a number of preliminar experiments, we set $\%_v$ to 50% for *Cat Food*, 10% for Herbs Cartons, and 5% for *Magazine Inserts*. Observe that TS provides the best results, and that both TS and HC provides better results in the

Table 3. Results for Tabu Search (20 runs per instance).

Problem	Number of Templates	Templates	Pressings	Overall Desv.	T.Desv
Cat Food	2	[0,0,0,0,0,2,7]	157143		
		[1,1,1,2,2,2,0]	250000	0.80	-3.85/ 1.79
	3	[0,1,1,0,0,0,7]	7143		
		[0,0,0,0,0,2,7]	150000		
		[1,1,1,2,2,2,0]	250000	0.14	-1.10/ 0.84
Hb.Cartons	2	[0,0,0,0,0,0,0,0,0,1,1,1,1,1,1, 1,2,1,1,2,2,2,2,1,6,2,2,2,5,5]	16400		
		[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,2,2,3,3,3,3,3]	66000	3.02	-8.44/ 10.00
	3	[0,0,2,2,2,2,2,2,2,0,0,0,0,3,3, 3,0,3,3,0,1,0,1,2,2,2,2,2,1,0]	3274		
		[0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1, 1,2,1,1,2,2,2,2,1,2,2,2,2,5,9]	17091		
		[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,2,3,3,3,3,3,2]	63090	0.73	-2.73/ 8.08
Mz.Inserts	3	[0,0,0,2,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,1, 1,0,0,0,0,0,0,1,2,2,1,2,0,3,1,1,0,0,0,0,3,3,2,6,3]	38250		
		[0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1, 1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,0,2]	150000		
		[1,1,1,0,0,1,2,2,0,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,2,0, 0,1,1,1,1,1,1,1,3,0,1,1,2,0,1,1,2,2,2,2,0,0,0,3,0]	50000	4.00	-10.00/ 42.86
	4	[0,0,0,0,1,1,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1, 0,0,1,2,2,1,2,2,2,2,2,1,0,3,0,0,3,0,2,1,1,0,0,0,3]	27750		
		[0,0,0,0,0,0,0,1,1,0,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1, 2,2,0,1,1,0,0,0,0,0,0,0,0,2,0,2,2,0,2,1,0,2,1,4,4,0]	38250		
		[0,0,0,0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 0,0,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,2,0,2,2]	111750		
		[1,1,1,1,1,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 2,2,1,2,2,1,1,1,1,1,2,1,1,1,1,1,1,1,0,1,0,4,0,2]	55000	2.59	-8.33/ 10.00

instances *Cat Food* and Herbs Cartons but worse in the instance *Magazine insert* with 4 templates than those given in [8]. The results for the GA are provided in Table 4. The GA matches the results obtained by the LS methods in the smaller instances but outperforms them in the most complex scenario (i.e., em Magazine Insert) producing here results close to those reported by Smith and Proll.

5 Conclusions and Future Work

The application of metaheuristics to the template design problem has resulted in very encouraging and positive results. An empirical comparison of three different techniques (i.e., a hill climbing method, a tabu search algorithm, and a genetic algorithm) has been conducted in an incremental scenario. The results of the population-based approach are better than those of the LS approaches, and get very close to (and actually in two out of three instances improve) the results reported in the literature for this problem, without requiring manual intervention as in [8].

We believe that further improved results can be attained with a hybrid proposal combining different algorithmic approaches, i.e., a memetic combination of HC/TS and GAs [5]. The form of this combination is an issue of further work.

Acknowledgements This work is supported by project NEMESIS (TIN-2008-05941) of the Spanish Ministerio de Ciencia e Innovación, and project TIC-6083 of Junta de Andalucía.

Table 4. Results for Genetic Algorithms (20 runs per instance).

Problem	No. Template	Templates	Pressings	Overall Desv.	T.Desv
Cat Food	2	[1,1,1,2,2,2,0]	250000		
		[0,0,0,0,0,2,7]	157143	0.80	-3.85/ 1.79
	3	[1,1,1,2,2,2,0]	250000		
		[0,0,0,0,0,2,7]	150000		
		[0,1,1,0,0,0,7]	7143	0.14	-1.10/ 0.84
Hb.Cartons	2	[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,2,2,3,3,3,3,3]	66000		
		[0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1, 1,1,1,2,2,2,2,2,1,6,2,2,2,5,5]	16400	3.02	-8.44/ 10.00
	3	[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,2,2,3,3,3,3,3]	62944		
		[0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1, 1,2,1,1,2,2,2,2,1,6,2,2,2,5,5]	17348		
		[0,0,2,2,2,2,2,2,2,0,0,0,3,3, 3,0,3,3,1,0,1,1,2,0,2,2,2,1,1]	3236	0.76	-2.36/ 8.49
Mz.Inserts	3	[0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]	150000		
		[1,1,1,1,1,1,2,0,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 1,0,1,1,0,1,1,1,1,1,1,1,2,1,1,1,1,1,0,1,1,2,2,3,3]	54500		
		[0,0,0,0,1,1,0,3,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1, 0,1,0,0,1,0,0,1,1,1,1,1,0,1,1,2,2,1,3,2,2,0,4,2,3]	30500	3.65	-9.75/ 10.00
	4	[0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]	140000		
		[1,1,1,1,1,1,2,0,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 1,0,1,1,0,1,1,1,1,1,1,1,2,1,1,1,1,1,0,1,1,2,2,3,3]	55000		
		[0,0,0,0,1,1,0,3,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1, 0,1,0,0,1,0,0,1,1,1,1,1,0,1,1,2,2,1,3,2,2,0,4,2,3]	30000		
		[0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0, 0,3,0,0,3,2,2,0,0,1,0,3,0,3,1,0,0,2,2,1,2,2,1,1,1]	10000	1.41	-6.00/ 10.00

References

1. Bäck, T., Fogel, D., Michalewicz, Z.: Evolutionary Computation 1: Basic Algorithms and Operators. Institute of Physics Publishing (2000)
2. Berkelaar, M., Eikland, K., Notebaert, P.: `lp_solve`, open source (mixed-integer) linear programming system (2004), http://lpsolve.sourceforge.net/5.0/
3. Glover, F.: Tabu search – part I. ORSA Journal of Computing 1(3), 190–206 (1989)
4. Glover, F.: Tabu search – part II. ORSA Journal of Computing 2(1), 4–31 (1989)
5. Moscato, P., Cotta, C.: A modern introduction to memetic algorithms. In: Gendrau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics (2nd Edition), International Series in Operations Research and Management Science, vol. 146, pp. 141–183. Springer-Verlag, New York (2010)
6. Prestwich, S.D., Tarim, S.A., Hnich, B.: Template design under demand uncertainty by integer linear local search. International Journal of Production Research 44(22), 4915–4928 (November 2006)
7. Proll, L., Smith, B.: ILP and constraint programming approaches to a template design problem. INFORMS Journal on Computing 10, 265–275 (1997)
8. Proll, L., Smith, B.: ILP and constraint programming approaches to a template design problem. Tech. Rep. 97.16, School of Computer Studies – Research Report Series (1998)
9. Syswerda, G.: Uniform crossover in genetic algorithms. In: Schaffer, J. (ed.) 3rd International Conference on Genetic Algorithms. pp. 2–9. Morgan Kaufmann, San Mateo, CA (1989)
10. Uppsala, P.F., Flener, P., Frisch, A.M., Hnich, B., Kzltan, Z., Miguel, I., Walsh, T.: Matrix modelling. In: CP-01 Workshop on Modelling and Problem Formulation. International Conference on the Principles and Practice of Constraint Programming (2001)