Discovering Beneficial Cooperative Structures for the Automated Construction of Heuristics

Germán Terrazas, Dario Landa-Silva and Natalio Krasnogor

Abstract The current research trends on hyper-heuristics design have sprung up in two different flavours: heuristics that choose heuristics and heuristics that generate heuristics. In the latter, the goal is to develop a problem-domain independent strategy to automatically generate a good performing heuristic for specific problems, that is, the input to the algorithm are problems and the output are problem-tailored heuristics. This can be done, for example, by automatically selecting and combining different low-level heuristics into a problem specific and effective strategy. Thus, hyper-heuristics raise the level of generality on automated problem solving by attempting to select and/or generate tailored heuristics for the problem in hand. Some approaches like genetic programming have been proposed for this. In this paper, we report on an alternative methodology that sheds light on simple methodologies that efficiently cooperate by means of local interactions. These entities are seen as building blocks, the combination of which is employed for the automated manufacture of good performing heuristic search strategies. We present proof-of-concept results of applying this methodology to instances of the well-known symmetric TSP. The goal here is to demonstrate *feasibility* rather than compete with state of the art TSP solvers. This TSP is chosen only because it is an easy to state and well known problem.

Germán Terrazas
ASAP Group, School of Computer Science
University of Nottingham, UK
e-mail: gzt@cs.nott.ac.uk

Dario Landa-Silva
ASAP Group, School of Computer Science
University of Nottingham, UK
e-mail: jds@cs.nott.ac.uk

Natalio Krasnogor
ASAP Group, School of Computer Science
University of Nottingham, UK
e-mail: nxk@cs.nott.ac.uk

1 Introduction

A *hyper-heuristic* is a search methodology that selects and combines heuristics to generate good solutions for a given problem. To investigate on the design of hyper-heuristics is important because they provide a problem independent level of abstraction for the automatic generation of good performing algorithms. Given a computational search problem and a set of simpler heuristics, hyper-heuristics contribute with a methodology for the manufacture of heuristic capable of producing high quality solutions when applied to the problem in hand. We consider that developing a systematic procedure in which beneficial entities are identified and combined for the automated manufacture of good performing heuristics is a suitable approach. The purpose of this paper is then to propose a method for the automated construction of heuristic search strategies in terms of simpler heuristic building blocks which cooperate efficiently. Our methodology has three main stages: pattern-based heuristics generation, cross validation and template-based heuristics distilling. In the following, Section 2 gives a brief introduction to hyper-heuristics and the context of our investigation. Section 3 expands on the proposed approach giving details of the model components and the methodology. After that, experiments and results are presented and discussed in Section 4. Finally, conclusions and further work are the subject of Section 5.

2 Heuristics Design

Hyper-heuristics are defined as search methodologies that select and combine low-level heuristics to solve hard computational search problems [6, 16]. The general aim of a hyper-heuristic is to manufacture unknown heuristics which are fast, well performing and widely applicable to a range of problems. During the process of fabrication, hyper-heuristics receive feedback from the problem domain which indicates how good the chosen heuristics for solving the problem in hand, hence driving the search process. Hyper-heuristics do not violate the no-free-lunch theory which indicates that over all problems, no algorithm performs better than another. Studying novel approaches for the development of hyper-heuristics is important since they are domain-independent problem strategies that operate on a space of heuristics, rather than on a space of solutions, and rise the level of generality on automated problem solving. Hyper-heuristics have been employed for solving search and optimisation problems such as bin-packing [17, 4], timetabling [14], scheduling [9, 8] and satisfiability [2] among others. For detailed reviews of hyper-heuristics and their applications, please refer to [16, 13, 7].

The automated manufacture of heuristic search strategies by means of hyper-heuristics has received increasing attention in the last ten years or so. Recent investigations have sprung up in two main different directions of hyper-heuristics: 1) heuristics that choose heuristics and 2) heuristics that generate heuristics. In the first case, a learning mechanism assists the selection of low-level heuristics according to

their historical performance during the search process, e.g. [8]. In the second case, the focus is on searching components that once combined generate a new heuristic suitable for the problem in hand. For example, approaches based on genetic algorithms [9] and genetic programming have been proposed for the automated generation of heuristics [5, 11]. From an engineering point of view, the already existent approaches are defined in terms of the architecture established by the underlying meta-heuristic which sometimes brings unsuspected difficulties such as the correct modelling of solutions or parameters tunning. Hence, the construction of well performing heuristics in terms of low-level heuristics which efficiently cooperate by means of local interactions is an interesting route for developing a new alternative within the second flavour of hyper-heuristics. Our interest lays on the identification of beneficial cooperative structures, the combination of which give rise to a specification for the automated manufacture of good performing heuristic strategies for a given combinatorial optimisation problem.

3 Proposed Approach

Given a set of instances of a combinatorial optimisation problem Π, we propose a methodology composed of pattern-based heuristics generation, cross validation and template-based heuristics distilling. Each stage is associated to a dataset generated from the optimisation problem in hand whilst the output of the methodology is a template to be employed for the manufacture of good performing heuristics. Fig. 1 depicts the methodology and its components.

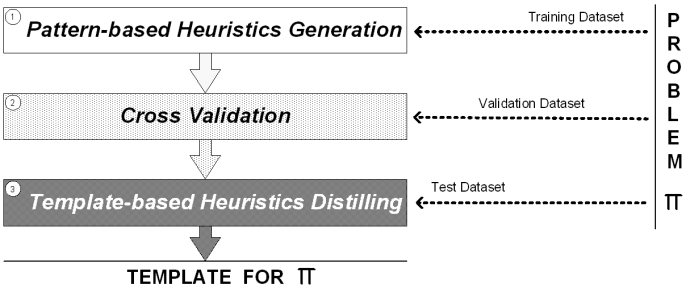

Fig. 1 Schematic representation of the proposed methodology with its three stages, their associated datasets and the achieved template for the problem in hand.

In the *pattern-based heuristics generation*, an input dataset is employed to train randomly generated sequences of low-level heuristics (high-level heuristics). This training aims at generating proficient high-level heuristics, the common constituents of which are expected to produce high quality solutions when applied to a given instance of the problem in hand. The research question in this stage is:

Given a set of high-level heuristics, is it possible to generate common combinations of low-level heuristics ? If yes, how do they look like and how reliable are these combinations ?

In order to address the first question, a process that spots common combinations of low-level heuristics (patterns) and constructs pattern-based heuristic is employed. The goal of the *cross validation* is then to assess the performance of the constructed pattern-based heuristic over a validation dataset comprising similar instances of the combinatorial optimisation problem in hand. Thus, the question in this stage is:

What is the performance of a pattern-based heuristic when applied to a set of different problem instances ?

The goal of the *template-based heuristics distilling* stage is to discover cooperative and efficient low-level heuristics (building blocks) among several pattern-based heuristics. These building blocks are expected to give rise to a template from where better than average heuristics could be drawn. Here, an extra dataset is employed to test the performance of the constructed heuristics. The question in this stage is:

Is it possible to distill a template in terms of building blocks of heuristics ? If yes, how is the performance of the template-based heuristics when applied to a set of different problem instances ?

The above methodology is expected to deliver a procedure for the automated construction of effective and efficient heuristic search strategies.

4 Methods and Results

This section presents the findings obtained by the above methodology. The chosen combinatorial optimisation problem is the widely known symmetric Traveling Salesman Problem (TSP). The TSP instance considered here is kroA100 which comprises 100 cities distributed in the Euclidean space. The objective value corresponding to the known optimum solution (shortest tour) for this instance is 21282 (see TSPLIB[1]). For each stage of our methodology, we generated five sets in the following systematic way. Each set is initialised with ten copies of the known optimum solution for kroA100. Each of this initial solutions is then 'disturbed' with n consecutive city swaps. In this way, setting n to 5, 25, 50, 75 and 100, a total of ten independently 'disturbed' tours per set are obtained.

We consider a high-level heuristic as a sequence made of low-level heuristics. The low-level heuristics for the TSP used here can be divided in two types: stochastic low-level heuristics and deterministic low-level heuristics. A low-level heuristic

[1] http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html

is stochastic if different or the same output tours are returned when applied to the same input tour. Contrary to this, a low-level heuristic is deterministic if the same output tour is returned when applied to the same input tour. In our case, *1-city insertion*, *2-exchange*, *arbitrary insertion* and *inver-over* are the stochastic low-level heuristics, whilst *2-opt*, *3-opt*, *OR-opt* and *node insertion* are the deterministic ones. These eight low-level heuristics were implemented as defined in [1, 3, 10, 15, 19] and operating in a hill climber style [12].

4.1 Pattern-based Heuristics Generation

4.1.1 Training Datasets

In this stage, each of the perturbed tours, labeled as $tkroA100_i^n$, $i = 0 \ldots 9$, $n = 5, 25, 50, 75, 100$, is independently considered for training. A sample of the training data, grouped by set (n), is listed in table 1 where the values indicate the percentage distance to the optimum from each perturbed tours.

Table 1 Three sample perturbed tours for each of the five training sets.

	$n = 5$	$n = 25$	$n = 50$	$n = 75$	$n = 100$
$tkroA100_0^n$	1.42669	4.25805	6.39869	7.01362	6.80147
$tkroA100_1^n$	1.27600	4.60262	6.46067	6.38215	6.59012
$tkroA100_2^n$	1.79926	4.13631	5.76585	6.75190	6.93252

$tkroA100_i^n$ is the i-th disturbed tour after applying n random swaps to *kroA*100 optimal tour.

4.1.2 Method

For a given disturbed tour ($tkroA100_i^n$), a set containing 500 high-level heuristics generated at random was created. Then, each of the 500 high-level heuristics was independently applied to the associated perturbed tour. In this context, an application is seen as a pipeline process in which the chain of processing elements is given by the sequence of low-level heuristics and the information to be processed is the disturbed tour. Thus, the low-level heuristics are applied one after another in the order in which they appear in the sequence and producing better or equal solutions at each step. To illustrate this process, Fig. 2 depicts how a high-level heuristic comprising 1-city insertion and 2-exchange is applied to a TSP instance.

In order to identify common combinations of low-level heuristics, the 500 high-level heuristics are then sorted according to the distance between the solution that their applications produce and the known optimum solution. The top five high-level heuristics are then selected and encoded as sequences of characters using ‘A’ to represent 1-city insertion, ‘C’ to represent 2-opt, ‘D’ to represent 3-opt, ‘E’ to represent

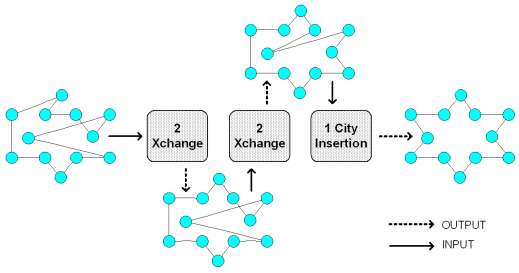

Fig. 2 A high-level heuristic in which successive applications of 1-city insertion and 2-exchange find the optimum solution for the Star of David tour.

OR-opt, 'T' to represent 2-exchange, 'F' to represent node insertion, 'G' to represent arbitrary insertion and 'H' to represent inver-over. Hence, in order to identify common combinations of low-level heuristics among the filtered sequence, we employ a multiple sequence alignment (MSA) method [18] over the encodings. For instance, Fig. 3 highlights in gray the common combinations found among the best five high-level heuristics generated for $tkroA100_2^{75}$.

```
1-HLH^75_3   ehhG--thHG------HHtG----------thhD---DhDh
2-HLH^75_3   ----aD---Gdadtdth----------------DddCD-D-
3-HLH^75_3   -----D--HG------H--Gaccchccaca--D--C----
4-HLH^75_3   ---G----HG------HH-G-------------D-------
5-HLH^75_3   ---Ge----G------H--G-------------D---D---
                |  |  ||      ||  |             |   ||  |
                |  |  ||      ||  |             |   ||  |
PBH^75_3     ---G-D--HG------HH-G-------------D--CD-D-
```

Fig. 3 Multiple sequence alignment of the top five heuristics. Capitals highlighted in gray indicate the common sequences of heuristics.

The results obtained by the MSA method reveal that there are indeed occurrences of common combinations, i.e. patterns of low-level heuristics, among the best ranked high-level heuristics. Thus, these findings give a positive answer to the research question stated for the first part of our methodology in Section 3.

From the resulting alignment, we construct a consensus sequence capturing and representing regions of similarity. We define this consensus sequence as a pattern-based heuristic (PBH_i^n) associated to a perturbed tour ($tkroA100_i^n$). The constructing procedure consists in copying the matching characters between two or more encodings into a new sequence from left to right and following the position in which they appear. For instance, Fig. 3 shows that PBH_3^{75} is the resulting pattern-based heuristic encoded as GDHGHHGDCDD, after combining the common patterns from the high-level heuristics 1-HLH_3^{75} to 5-HLH_3^{75}. Given that this new heuristic is built in terms of common combinations of low-level heuristics, its performance is then expected to be as good as (or better than) any of the top ranked. Notice that the length of the constructed heuristic varies according to the number of matches. Since this is related to the way in which the construction procedure is defined, alternative

methodologies to obtain the optimal common sequence are open to further investigation.

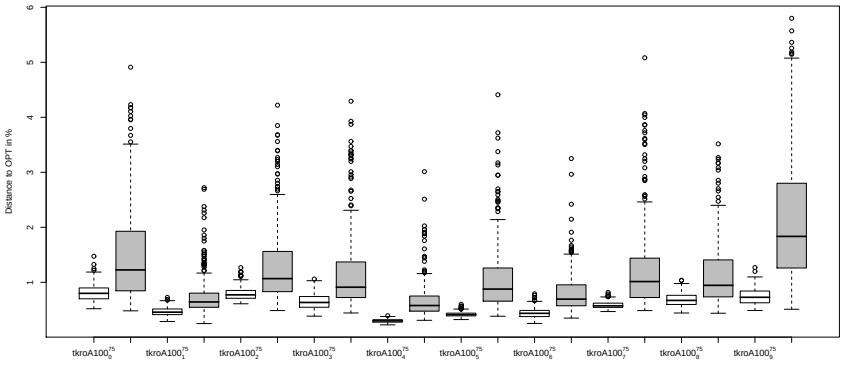

Fig. 4 Assessment of ten pattern-based heuristics resulting from independent sequence alignments. Each pair of boxplots summarises a vis-a-vis comparison between the performance of 300 copies of PBH_i^{75} and the performance of other 300 high-level heuristics when applied to $tkroA100_i^{75}$ for $i = 0 \ldots 9$.

In order to assess the reliability of the spotted patterns, we then proceed to evaluate the performance of PBH_i^n against a set of high-level heuristics (different than the initial ones) with the hope that, on average, the best tour improvements are obtained by the former. In order to do this, 300 copies of PBH_i^n are obtained and for each of them a new high-level heuristic equal in length is created. Each of these heuristics is then independently applied to $tkroA100_i^n$ a total of 10 times and the average percentage distance between the lengths of the resulting tours and the known optimum is considered as the measure of their performance. As an example, Fig. 4 shows the assessment of the 10 pattern-based heuristics obtained from the data set generated with $n = 75$.

According to the results, it is clear that the performance of pattern-based heuristics (white boxplots) is better in average than the performance of the non-pattern-based high-level heuristics (gray boxplots). These findings constitute a positive answer to the second research question stated in the first stage of the presented methodology, i.e. the identified common-sequences of heuristics are indeed reliable.

4.2 Cross Validation

4.2.1 Validation Dataset

The cross validation data are given in sets of ten perturbed tours $vkroA100_i^n$, $i = 0 \ldots 9$. A sample of the data, grouped by set (n), is listed in table 2 where the values indicate the percentage distance to the optimum from each perturbed tours.

Table 2 Three sample perturbed tours for each of the five validation sets.

	$n=5$	$n=25$	$n=50$	$n=75$	$n=100$
$vkroA100_0^n$	1.86490	5.38403	6.85800	6.92453	7.58471
$vkroA100_1^n$	1.72394	5.42246	6.13800	6.57452	6.69500
$vkroA100_2^n$	1.41001	3.76134	6.66469	6.85969	6.90264

$vkroA100_i^n$ is the i-th disturbed tour after applying n random swaps to $kroA100$ optimal tour.

4.2.2 Method

The goal of this stage is to perform a cross validation analysis in order to assess the performance of the pattern-based heuristics over a set of disturbed tours. Thus, for each combination of PBH_j^n and $vkroA100_i^n$, $i,j = 0\ldots9$, a total of 300 copies of PBH_j^n were obtained and, for each of the copies, a new high-level heuristic equal in length was created. Then, the heuristics are independently applied to the given $vkroA100_i^n$ a total of 10 independent times and the average percentage distance between the lengths of the resulting tours and the known optimum is considered as the measure of their performance. Fig. 5 shows the resulting assessment of a pattern-based heuristic, encoded as GDHGHHGDCDD, over the 10 perturbed tours belonging to the data set generated with $n = 75$.

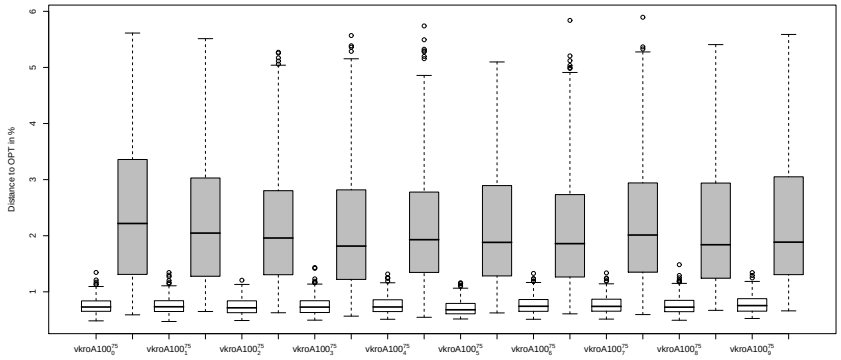

Fig. 5 Performance evaluation of a pattern-based heuristic across the perturbed tours belonging to the data set generated with $n = 75$. Each pair of boxplots summarises a vis-a-vis comparison between the performance of 300 copies of GDHGHHGDCDD and the performance of other 300 high-level heuristics when applied to $vkroA100_i^{75}$.

Clearly, the performances of pattern-based heuristics (white boxplots) are better in average than the performance of the ones generated for assessment (gray boxplots). These findings answer the research question estated in the second part of Section 3, revealing that a pattern-based heuristic is in general well performing when applied to a set of different problem instances. In addition, the similar level of performance observed among the white boxplots gives an indication that common low-level heuristics could be acting as building blocks among the PBH_j^n, $j = 1\ldots10$.

4.3 Template-based Heuristics Distilling

4.3.1 Test Dataset

The data used in this last stage comprise five sets listed, see Table 3 for a sample. Thus, for a given experiment, each of the ten perturbed tours $dkroA100_i^n$, $i = 0 \ldots 9$, belonging to a given set is independently employed for testing.

Table 3 Three sample perturbed tours for each of the five test sets.

Tour	$n = 5$	$n = 25$	$n = 50$	$n = 75$	$n = 100$
$dkroA100_0^n$	1.43750	4.00032	5.61831	6.34000	6.86100
$dkroA100_1^n$	1.12729	4.70731	6.44469	6.28794	6.69199
$dkroA100_2^n$	0.80584	4.01320	5.96786	6.57973	7.10008

$dkroA100_i^n$ is the i-th disturbed tour after applying n random swaps to $kroA100$ optimal tour.

4.3.2 Method

The purpose of this stage is to identify common building blocks of low-level heuristics among the PBH_j^n assessed in the second part of our methodology. These building blocks are employed to construct templates of heuristics, the instances of which are expected to show similar or better performance when solving any $dkroA100_i^n$. Hence, for each data set, we applied the MSA method over the encodings of PBH_j^n, $i, j = 0 \ldots 9$. For example, Fig. 6 highlights in gray building blocks among the ten pattern-based heuristics found for the data set of perturbed tours generated with $n = 75$.

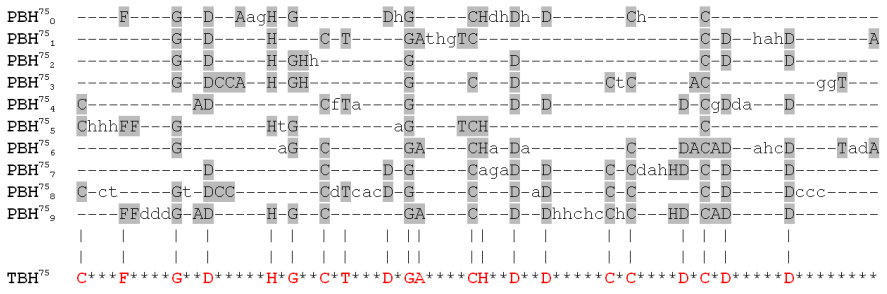

Fig. 6 Multiple sequence alignment of the pattern-based heuristics found for the data set generated with $n = 75$. Capitals highlighted in gray indicate the spotted common building blocks.

The resulting alignment reveals that there are common structures among the pattern-based heuristics. A template (TBH^n) is then constructed in terms of building

blocks. This procedure consists in copying the matching characters between three or more encodings into a new sequence from left to right and following the position in which they appear. In case no matchings are found or matchings occur only between two encodings, a wildcard character is placed in that position of the sequence. For instance, Fig. 6 shows TBH^{75} as the resulting template after combining the building blocks from the input pattern-based heuristics PBE_0^{75} to PBE_9^{75}.

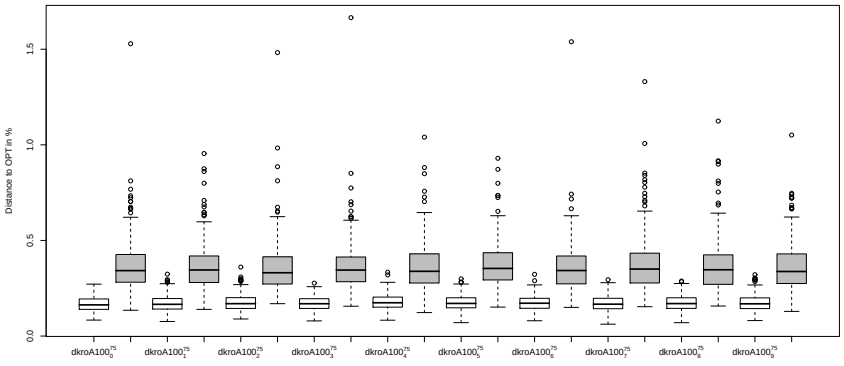

Fig. 7 Assessment of a template-based heuristic across a set of perturbed tours belonging to the data set generated with $n = 75$. Each pair of boxplots summarises a vis-a-vis comparison between the performance of 300 instances drawn from TBH^{75} and the performance of other 300 high-level heuristics when applied to $dkroA100_i^{75}$ for $i = 0 \ldots 9$.

For each $dkroA100_i^n$, a total of 300 different instances are drawn from the constructed template. During the instantiation process, building blocks are preserved and each of the wildcard characters is either removed or replaced with one of the eight low-level heuristics chosen at random. In order to assess the reliability of the building blocks, we compared the performance of the 300 instances against new 300 high-level heuristics expecting that, on average, the best tour improvements are obtained by the former. In this way, each of the heuristics is applied to the same perturbed tour a total of 10 independent times and the average distance between the lengths of the resulting tours and the known optimum is considered as the measure of performance. A representative outcome of the assessment is shown in Fig. 7 where the resulting assessment of the instances drawn from TBH^{75} when applied to the data set created with $n = 75$ is depicted.

The results of this stage demonstrate that it is certainly possible to define a template of building blocks of heuristics in terms of common structures identified among a set of pattern-based heuristics. This fact constitutes a positive answer for the first question established in the third part of our methodology. In addition, it is also shown that the performance of template-based heuristics (white boxplots) is on average better than the performance of the randomly generated high-level heuristics (gray boxplots), even though some of the high-level heuristics generated for comparison have outperformed the ones drawn from the template (see Fig. 7). Naturally, one of the reasons for this is that during the random generation, appropri-

ate combinations of low-level heuristics with more efficient local interactions could be generated (by chance). However, the template-based specification still brings a more robust and convenient way for the automated manufacture of good performing heuristic strategies to solve the problem in hand. All in all, the outcome of this assessment answers the last question of the proposed methodology. That is, the instances of such templates are always well performing when applied to any disturbed tour of a given data set.

5 Conclusions

In this paper, we proposed a novel approach for the automated design of heuristics following the rationale of hyper-heuristics which are heuristic methods to generate tailored heuristics for the problem in hand.

The proposed methodology consists of pattern-based heuristics construction, cross validation and template-based heuristics construction. As a proof of concept, we applied the methodology to instances of the symmetric TSP. On the one hand, our initial findings confirm that there are indeed common patterns of low-level heuristics among the top ranked high-level heuristics. These emergent recurrent structures were subject to a cross validation, the results of which proved them to be local search strategies beneficial to achieve good solutions when solving a symmetric TSP instance. On the other hand, the outcome achieved in the last part of our approach has resulted in a specification to automatically generate a family of heuristics capable of producing high quality solutions when applied to perturbed tours. In particular, these high performing heuristics are made of emergent building blocks extracted from the patterns seen in the first stage.

From a functional point of view, the building blocks achieved in the last stage are beneficial structures needed for the manufacturing of high quality solutions. When these key elements appear in combination with randomly chosen low-level heuristics, they seem to guide the search across the space of solutions. In other words, the local interactions contributed by the building blocks can be seen as artifacts that drive the optimisation process when applied to the combinatorial optimisation problem in hand. Likewise, the local interactions contributed by the randomly created low-level heuristics placed in an instance can be seen as artifacts that contribute with a variety of alternative paths for exploring the space of solutions during the optimisation process. Hence, both types of contributions seem to be properly orchestrated into an instance of a template-based heuristic.

To continue with our methodology, future work involves the extension of our approach to other instances of TSP as well as to different combinatorial optimisation problems. In addition, we also consider to explore alternative ways to generate the family of good performing heuristics in order to get a faster and less human-dependent way. This could be done for instance by means of grammatical inference where the encodings of the pattern-based heuristics would be the input to the

grammatical inference algorithm and the resulting grammar would be employed to generate a family of words encoding sequences of low-level heuristics.

References

1. G. Babin, S. Deneault, and G. Laporte. Improvements to the or-opt heuristic for the symmetric traveling salesman problem. *Journal of the Operational Research Society*, (58):402–407, 2007.
2. M. Bader-El-Den and R. Poli. A gp-based hyper-heuristic framework for evolving 3-sat heuristics. In *Genetic and Evolutionary Computation Conference*, pages 1749–1749. ACM, 2007.
3. J. Brest and J. Zerovnik. A heuristic for the asymmetric traveling salesman problem. In *6th Metaheuristics International Conference*, pages 145–150, 2005.
4. E. Burke, M. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic. In *9th International Conference on PPSN*, volume 4193, pages 860–869. Springer-Verlag, 2006.
5. E. Burke, M. Hyde, G. Kendall, and J. Woodward. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Genetic and Evolutionary Computation Conference*, pages 1559–1565. ACM, 2007.
6. E. K. Burke, E. Hart, G. N. Kendall, J. Newall, P. Ross, and S. Schulenburg. *Handbook of Meta-Heuristics*, chapter Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pages 457–474. Kluwer, 2003.
7. K. Chakhlevitch and P. Cowling. Hyperheuristics: Recent developments. In *Adaptive and Multilevel Metaheuristics*, volume 136, pages 3–29. Springer, 2008.
8. P. Cowling and K. Chakhlevitch. Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In *IEEE Congress on Evolutionary Computation*, pages 1214–1221. IEEE Computer Society, 2003.
9. P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *IEEE Congress on Evolutionary Computation*, pages 1185–1190. IEEE Computer Society, 2002.
10. N. Krasnogor and J. Smith. Memetic algorithms: The polynomial local search complexity theory perspective. *Journal of Mathematical Modelling and Algorithms*, 7:3–24, 2008.
11. M. Oltean and D. Dumitrescu. Evolving tsp heuristics using multi expression programming. In *4th International Conference on Computational Science*, volume 3037, pages 670–673, 2004.
12. E. Özcan, B. Bilgin, and E. Korkmaz. Hill climbers and mutational heuristics in hyperheuristics. In *9th International Conference on PPSN*, pages 202–211, 2006.
13. E. Özcan, B. Bilgin, and E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intell. Data Anal.*, 12(1):3–23, 2008.
14. N. Pillay and W. Banzhaf. A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem. *European Journal of Operational Research*, 197(2):482–491, 2009.
15. G. Reinelt. *The traveling salesman: Computational solutions for TSP applications*. Springer-Verlag, 1994.
16. P. Ross. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support*, chapter Hyper-heuristics, pages 529–556. Springer, 2005.
17. P. Ross, S. Schulenburg, J. Marín-Blázquez, and E. Hart. Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In *Genetic and Evolutionary Computation Conference*, pages 942–948. Morgan Kaufmann Publishers Inc., 2002.
18. J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing, 1997.
19. G. Tao and Z. Michalewicz. Inver-over operator for the tsp. In *5th International Conference on PPSN*, pages 803–812. Springer-Verlag, 1998.