Automatic Parameter Configuration for an Elite Solution Hyper-Heuristic Applied to the Multidimensional Knapsack Problem

Enrique Urra
Escuela de Ingeniería en Informática
Pontificia Universidad Católica de Valparaíso
Av. Brasil 2950, Valparaíso, Chile
enrique.urra.c@mail.pucv.cl

Daniel Cabrera-Paniagua
Escuela de Ingeniería Comercial
Universidad de Valparaíso
Pasaje La Paz 1301, Viña del Mar, Chile
daniel.cabrera@uv.cl

Claudio Cubillos
Escuela de Ingeniería en Informática
Pontificia Universidad Católica de Valparaíso
Av. Brasil 2950, Valparaíso, Chile
claudio.cubillos@ucv.cl

Gastón Lefranc
Escuela de Ingeniería Eléctrica
Pontificia Universidad Católica de Valparaíso
Av. Brasil 2950, Valparaíso, Chile
gaston.lefranc@gmail.com

***Abstract*—Hyper-heuristics are methods for problem solving that decouple the search mechanisms from the domain features, providing a reusable approach across different problems. Even when they make a difference regarding metaheuristics under this perspective, proposals in literature commonly expose parameters for controlling their behavior such as metaheuristics does. Several internal mechanisms for automatically adapt those parameters can be implemented, but they require extra design effort and their validation no necessarily is generalizable to multiple domains. Such effort is prohibitive for their practical application on decision-support systems. Rather than implementing internal adapting mechanisms, the exploration of automatic parameter configuration through external tools is performed in this work. A new hyper-heuristic implementation based on a elite set of solutions was implemented and automatically configured with SMAC (Sequential Model-Based Algorithm Configuration), a state-of-art tool for automatic parameter configuration. Experiments with and without automated configuration are performed over the Multidimensional Knapsack Problem (MKP). Comparative results demonstrate the effectiveness of the tool for improving the algorithm performance. Additionally, results provided insights that configurations applied over subsets of instances could provide better improvements in the algorithm performance.**

***Index Terms*—hyper-heuristics, automated algorithm configuration, multidimensional knapsack problem, sequential model-based algorithm configuration.**

I. Introduction

In the optimization scenario, numerous algorithms have been proposed to solve a diversity of problem domains. These solvers range from pretty specific and simple techniques to complex algorithms with architectures capable of addressing numerous problems, or families of problems. Considering the number of different scenarios that a single solver can face, it is common to *parametrize* their behavior in order to achieve a better adaptability and flexibility on the solving process. This parametrization involves, however, an important trade-off regarding their empirical evaluation. The more larger the number of parameters is, the more difficult is to provide generalizable performance comparisons of the involved solvers. Parameters can adopt multiple values, even under a continuous space, making hard to select a good set of values with scientific rigor under a manual process. Because of this, in the latter years there have been a growing interest on methodologies and tools for effectively *automate* the process of selecting values for solver parameters. Even such process has been conceptually generalized to the design automation of algorithms from selectable building blocks [1], [2].

In the context of heuristic methods, it is common to provide parameters for customizing their behavior. Even when several tendencies point to include mechanisms to dynamically adjust parameter values through complementary adaptive behavior components, their introduction involves additional effort that sometimes is difficult to achieve. For example, first versions of hyper-heuristic solvers [3], [4], [5] included a parametrized operator named *Choice-Function*, whose configuration allowed to change the tendencies of exploration and exploitation during the search. These parameters were after replaced by an adaptive mechanism that change them during the search. Even if such mechanism was successful for the domains in which they were tested, it is difficult to predict if such efficiency will be replicated on totally different domains and testing conditions. This issue is also common for metaheuristics [6], [7], [8]. Even when they are defined as conceptual frameworks, they commonly include at a higher-level specification several parameters that are concretely defined when the framework is adapted to a specific problem domain. The adaption of such parameters to the specific problem at the same time nothing says about how different adaptations to different domains will perform.

Rather than proposing adaptive mechanisms for heuristic parameters, in this work, the exploration of automatic param-

eter configuration through external tools is performed. Even when this approach can result a natural way to proceed, it is rarely applied in literature. Particularly, hyper-heuristics are considered as the solving technique in this work. To the knowledge of the authors, no approach in literature uses external automated configuration for hyper-heuristic parameters, even when numerous parametrized solvers have been proposed in literature. A new hyper-heuristic implementation based on a elite set of solutions was implemented, which exposes several parameters. It is automatically configured with *SMAC* (Sequential Model-Based Algorithm Configuration) [9], a state-of-art tool for automatic parameter configuration that has been proven to provide extensive improvement on highly parameterized algorithms. Experiments with and without automated configuration are performed over the Multidimensional Knapsack Problem (MKP), a well-known combinatorial problem from literature [10]. Results are compared and they demonstrated the effectiveness of the tool when improving algorithm performance. Additionally, results provided insights that *partial configurations*, i.e., configurations applied over subsets of instances rather than all of them, could provide better improvements in the algorithm performance. It is expected that this work will contribute to demonstrate the usefulness of using automated tools for complex heuristic methods such as hyper-heuristics, and to incentive their use for improving the validity of experimental evaluation related to heuristic methods. Also, this could improve their practical application to decision-support systems, as the parametrization is focused more on the adaption to user needs rather than technical details of the internal solver used.

This work is presented in the following sections. Section II addresses the hyper-heuristic technology, and provides a general description of the elite set hyper-heuristic implemented during this work. Section III provides several background on the MKP. In Section IV, a general view on automated algorithm configuration is presented, which includes several practical considerations in the use of SMAC. Section V is devoted to describe the performed experiments and review the gathered results through an incremental approach. Finally, Section VI presents several conclusions of this work.

II. Hyper-heuristics

Hyper-heuristics fall under the category of heuristic methods, i.e., behavioral mechanisms for problem solving, ranging from simpler algorithms to complex and intelligent learning techniques. They emerge as an alternative to other well-known heuristic methods, e.g. metaheuristics [7]. The main difference between both is related to their design approach. Metaheuristics are algorithmic frameworks that define abstract solution-search mechanisms for problem solving purposes. Because of this abstraction, their applicability to problems lies on the detail with which they are adapted to a particular problem. The main consequence of this adaption is a tight coupling between the search operations with problem domain features, with a deep specialization on the latter. On the other hand, hyper-heuristics design does not depend directly in adaption, but in modularity and extensibility. The main search operations are isolated in a specific layer, the hyper-heuristic itself, and the problem domain is implemented independently. This allows to achieve an effective decouple of the solving mechanism regarding problem domain features, with a trade-off on problem specialization.

A hyper-heuristic has been formally defined as the process of *using heuristics to choose heuristics*, for solving a problem in hand [11]. However, Burke et al. proposed a new definition which is based on the current research trends in hyper-heuristics [12]: A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational search problems. The use of the word "generation" involves a completely different approach through which new heuristics can be generated by a hyper-heuristic method.

A *selection hyper-heuristic* is commonly structured as Figure 1 shows. There is a high-level in where the hyper-heuristic

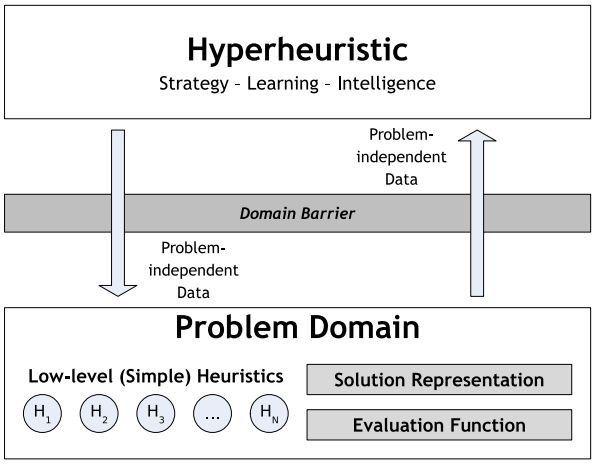

Fig. 1. The generic hyper-heuristic framework.

itself resides and it is outside the problem domain. Also, there is a low-level, which can be considered the problem domain itself. At this level, a collection of *low-level heuristics* resides, which use rich problem knowledge to operate. The main features of these low-level heuristics are their simplicity and specialization degree. For example, they can be swap, add and drop moves from a specific problem context. Also, there is a *domain barrier* between the hyper-heuristic and the low-level heuristics, which prevents to pass problem knowledge from the low-level to the high-level. Therefore the hyper-heuristic only knows about the existence of the low-level heuristics as executable entities. The behavior which they use to perform some operation in the problem domain is completely encapsulated.

A. An Elite Solution Hyper-heuristic Extension

Different kinds of behavior can be implemented in the hyper-heuristic layer, ranging from deterministic or non-learning [5], [13], [3], [14] to more-intelligent, learning-based

ones [4], [15], [16], [17]. In this work, we propose a new mechanism based on a simple learning approach to avoid premature convergence towards local-optima. It consists on an *elite solution set* extension (ESE-HH), i.e., a memory of good solutions found through the search. Limited in size by a parameter λ, this list can be updated on each iteration with the solution generated by a selected low-level heuristic. If the search is stuck in some local optima, i.e., a number of τ iterations have passed since the last global improvement, an element of the elite set can be used to replace the current solution.

The code of ESE-HH is shown in Listing 1. The process is

```
1  function hyperheuristic()
2    initialize problem domain and hyper-heuristic
3    run H_init and store its result in S_accepted
4    set S_best = S_accepted, S_candidate = null
5    set g_i = 0, E = []
6    while stop conditions have not been met
7      S_candidate = heuristic_selection(S_accepted)
8      update_elite_set(S_candidate, E, λ)
9      if move_acceptance(S_candidate) = true
10       set S_accepted = S_candidate
11     end if
12     if S_candidate > S_best
13       set S_best = S_candidate, g_i = 0
14     else
15       set g_i = g_i + 1
16       if g_i >= τ
17         S_elite = select_elite_solution(E)
18         set S_accepted = S_elite, g_i = 0
19       end if
20     end if
21   end while
22   return S_best
23 end function
```

Listing 1. Pseudocode of the elite solution set extension.

initialized between lines 2 and 5, which include the generation of an initial solution, the init of the non-improving global iterations g_i and the initialization of the elite set E. The main iteration is performed between lines 6 and 21. In line 7, a *heuristic selection* operator is executed, i.e., an operator that runs several low-level heuristics with some criteria and generates a new solution from the current accepted one. The elite solution set is updated in 8 with the new generated solution. In line 9, the *move acceptance* operator is executed, which decides if the new generated solutions will be keep as the current accepted one. In line 13, if a new best solution is found, the g_i value is restarted. Otherwise, the latter is increased and, in line 18, if the τ parameter is reached, the current accepted solution is directly replaced by an elite one, restarting g_i as well. At the end (line 22), the best solutions is returned as result.

Regarding the `select_elite_solution` function two different approaches were considered:

- A *random* selection approach, in where all solutions in the elite list have an uniform probability of being selected to replace the current accepted solution.
- A *roulette wheel* selection approach, in where each elite solution in the list has a probability to be selected according to the proportion of its fitness among the fitness of all elite solutions. An additional *amplification* parameter σ is used in this selection schema to amplify the proportion of each solution. Let be p_i the probability of the i-solution in the elite set to be selected with the roulette wheel. Then, p'_i is the amplified probability, calculated with the following formula dependent of σ:

$$p'_i = p_i \cdot \Big(1 + p_i \cdot \sigma\Big),\ \sigma \geq 0 \tag{1}$$

With this amplification, solutions with higher probability will increase their chances regarding the ones with lower probability. The σ can help to establish more notorious differences between selection chances of solutions in the elite list.

The use of the random or roulette wheel in the ESE is configurable by an additional parameter. Note that σ will be available only if the roulette wheel selection is used.

III. The Multidimensional Knapsack Problem

The Multidimensional Knapsack Problem (MKP), also known in literature as the Multidimensional 0-1 Knapsack Problem, is a generalization of the 0-1 Knapsack problem (KP) [10], a well-known problem from the operations research. In the KP, there is a set of knapsacks with limited space, and a set of items to be selected. Each item covers a portion of the knapsack space and provides a specific profit. The goal is to select a set of items for which the total profit is maximized and space constraints are respected. The MKP is named "multidimensional" because, rather than being constrained to a single knapsack, there are multiple knapsacks to be filled at the same time. Each item to be selected can fill a different quantity on each knapsack. However, when an item is selected, all knapsacks are filled with such item.

The MKP is mathematically formulated as follows. There is a set of items $I = \{1, 2, ..., n\}$ and a set of resources (knapsacks) $R = \{1, 2, ..., m\}$. Let be b_i the capacity of the resource i and a_{ij} the consumption of the item j on the resource i. Consider a decision variable x_j, which is 1 if the item j is selected, and 0 otherwise. Finally, let be c_j the profit of selecting the item j. Then, the mathematical formulation of the problem is as follows:

$$\max \sum_{j=1}^{n} c_j \cdot x_j \tag{2}$$

$$\text{subject to} \sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i,\ \forall i \in R \tag{3}$$

There is extensive literature regarding the MKP problem. Only some are mentioned in this section. In the metaheuristic context, perhaps one of the most well known studies is the GA developed by Beasley et al. [18]. Apart from the implemented mechanisms, this study has relevance because a set of standard benchmark sets were generated on it. Further literature extensively used such benchmarks for performance comparison. Other techniques include an Harmony Search [19], Memetic

Lagrangian Heuristic [20] and greedy approaches [21]. The hyper-heuristic literature also includes several approaches for solving the MKP. The well known choice-function has been modified in combination with a late acceptance strategy, i.e., an specific move acceptance operator, for its application to the MKP [22]. A different approach compares mechanisms to control crossover parameters that affect the behavior on both the high and the low levels of the hyper-heuristic framework, using the MKP as case study [23]. Also, genetic programming-based hyper-heuristics have been implemented for the MKP problem [24].

IV. Automatic Algorithm Configuration

It is common for optimization mechanisms and tools to expose several parameters for fine-tunning their behavior under different application conditions. The process of defining the values of such parameters is lengthly and complex, particularly when the number of exposed parameters is higher. Manually tuning such values could importantly harm the validity of empirical studies performed over parametrized algorithms. Automating this process had been a focus of attention in the latter years. Several alternatives and tools exist for these purposes in literature. *ParamILS* [1], [2] corresponds to an automatic parameter configuration framework based on an Iterated Local Search (ILS) mechanism. Important speedups were obtained in different domains by applying this tool on algorithms that expose an important number of parameters. A more recent and better alternative is *SMAC* (Sequential Model-Based Algorithm Configuration) [9], an automated parameter framework that uses a fitting model strategy named Sequential Model-based Optimization (SMBO) for finding a good parameter configuration. SMAC has been empirically proved to outperform the speedups obtained by other techniques such as ParamILS for highly parametrized algorithms, representing one of the current state-of-art techniques in algorithm configuration.

SMAC is a state-of-art tool regarding automated algorithm configuration, particularly when different types of parameters are exposed by algorithms. It can automatically configure discrete, continuous and categorical parameters. This allows to find not only numerical parameter configurations but to evaluate if it is better or worst to enable or disable specific algorithm components. This is particularly useful for extensions such as ESE-HH, because parameters can be defined to enable or disable them. SMAC also supports dependent parameters, i.e., child parameters those values are used only when other parent parameters are active/inactive, or then the parent parameters have specific values. In this way, when some extension is disabled, its parameters can be disabled at the same time.

In this work, SMAC is used to configure the parameters of the ESE-HH extension and to evaluate if it is worth to enable such extension when a hyper-heuristic is applied to the MKP problem. Several practical issues were considered regarding the application of SMAC:

- A relevant element regarding SMAC usage over any parametrized algorithm is the implementation of a *wrapper*, i.e., an executable component that can i) receive inputs from SMAC, ii) run the algorithm to configure based on such inputs, and iii) provide outputs in a standard format that can be used by SMAC as information of the configuration process. As a part of this research, several wrappers where implemented, which allowed to connect SMAC with the implemented hyper-heuristics.
- In the experiments, SMAC would be executed with different configurations of allowed time for the hyper-heuristic to run, which is known as *cap-time*. With this, there should be different options of configuration, considering that such a factor is relevant: if a lower value is selected, the configuration is weighted to small instances, and a bigger value may spend too many time in configuration. Also, a *wall-clock time* should be considered, i.e., the total time allowed for configuration.
- SMAC uses two different instances groups: one is for *training*, over which the configuration is run, and the other is for *testing*, over which the obtained configuration is validated. Optimally, there should be no intersection between both groups, however, the test group should represent in some degree the training group, at least on instance features.

V. Experiments, Results and Discussion

An incremental approach to automatically configure ESE-HH with SMAC was performed during this research. To carry on a comparative approach, three different hyper-heuristic variants were tested:

- **D-HH**. Only non-learning operators were used: *greedy* heuristic selection and *improve or equals* move acceptance [3]. The former runs all low-level heuristics and keeps the result of the best performing one. The latter accepts solutions only if the quality is better or equal than the current solution. This approach is similar to an implementation previously used with a transportation problem [25].
- **ESE-HH**: Corresponds to the D-HH with the ESE-HH enabled.
- **AC-HH**: Corresponds to the hyper-heuristic automatically configured with SMAC.

Particularly for AC-HH, a parameter e_i will be 1 if ESE-HH is enabled or 0 otherwise. If enabled, the following parameters are enabled at the same time: *Elite set max. size* (λ), *Convergence tolerance* (τ) and *Elite selection procedure* (s_p), which can be *random* or *roulette wheel*. If s_p is *roulette wheel*, the parameter *Roulette wheel amplifier* (σ) is enabled as well. The default values for e_i, λ, τ, s_p and σ are 100, 20, *roulette wheel* and 100.0 respectively, which are used in ESE-HH and as start configuration point of AC-HH. For all hyper-heuristic runs, a maximum of 200000 iterations were allowed. For each execution, 20 repetitions were executed to calculate average values.

Regarding benchmark instances, the complete set contains 270 different instances, divided in groups of 10. Each group is a combination of three instance features: *number of constraints* (knapsacks, can be 5, 10 and 30), *number of variables* (items, can be 100, 250 and 500), and *tightness ratio*, i.e., the proportion between a resource capacity and the sum of all item consumptions for such resource (can be 0.25, 0.5 and 0.75). The standard way to measure solution performance in research related to MKP is to use a *percentage gap* between solutions and the optimal value of the LP-relaxed problem. Let s_i be the objective function result for the instance i and o_i is the LP-optimum of such instance. Then, the percentage gap g_i is calculated with the following formula:

$$g_i = 100 \times \frac{o_i - s_i}{o_i} \tag{4}$$

This gap allows to compare results independently of instance features, or to group instances according to several features for comparison, e.g., results for a specific number of constraints or a specific number of variables.

A. Results with Initial Configuration Setup

In a first approach to automatic configuration, the following SMAC setup was considered: a *cap-time* of 480 seconds, a wall-clock time of 2 days, and a ratio of 90%-10% for training and testing instances, respectively (i.e., in each group of 10 heterogeneous instances, one was picked for testing and the remaining for training). Table I shows the results of this configuration along with the default hyper-heuristics variants described before. The first three columns show features of the instances mentioned before: number of constraints (cons), number of variables (vars) and tightness ratio (tigh). The following columns are the gap means for each hyper-heuristic. Although ESE-HH outperformed D-HH in the average gap, AC-HH not improved ESE-HH as one could expect. Actually, their results were sightly worser than the ESE-HH ones. Different aspects of SMAC configuration were analyzed as possible reasons for this. First, when using a *cap-time* of 480 for the automated configuration, and not using the same limit on tests, it was highly likely to worsen comparability, as the provided configuration may not perform in the same way on both. Second, the *wall-clock time* used (2 days) could be too low for SMAC to perform enough exploration in the parameter space. And finally, the number of testing instances (10%) were considered as a possible source of poor performance for adequate measuring of the configuration provided by *SMAC*.

TABLE I
FIRST RESULTS OBTAINED FOR THE DIFFERENT COMBINATIONS OF HYPER-HEURISTICS MODELS, ON EACH GROUP OF INSTANCE FEATURES FOR THE MKP. THE RESULTS ARE SHOWN AS THE AVERAGE VALUE (OVER 20 REPETITIONS) OF THE GAP (THE LOWER, THE BETTER).

			Model		
cons	**vars**	**tigh**	D-HH	ESE-HH	AC-HH
5	100	0.25	0.01805	0.01172	0.01236
		0.5	0.00969	0.00616	0.00635
		0.75	0.00639	0.00418	0.00432
	250	0.25	0.01816	0.01155	0.01228
		0.5	0.01055	0.00689	0.00687
		0.75	0.00572	0.00379	0.00390
	500	0.25	0.02168	0.01810	0.01712
		0.5	0.01214	0.01193	0.01062
		0.75	0.00705	0.00644	0.00613
10	100	0.25	0.03053	0.02033	0.02205
		0.5	0.01645	0.01055	0.01112
		0.75	0.00851	0.00604	0.00614
	250	0.25	0.03011	0.01603	0.01727
		0.5	0.01520	0.00869	0.00901
		0.75	0.00819	0.00479	0.00510
	500	0.25	0.03256	0.02189	0.02160
		0.5	0.01673	0.01196	0.01168
		0.75	0.00945	0.00620	0.00669
30	100	0.25	0.04477	0.03568	0.03784
		0.5	0.02191	0.01618	0.01728
		0.75	0.01258	0.00951	0.01036
	250	0.25	0.03725	0.02343	0.02595
		0.5	0.01725	0.01127	0.01201
		0.75	0.00886	0.00586	0.00643
	500	0.25	0.03375	0.02319	0.02441
		0.5	0.01575	0.01078	0.01140
		0.75	0.00858	0.00588	0.00650
		AVG	0.01770	0.01219	0.01269

B. An Improved Configuration Setup

Considering the issues described before, a new hyper-heuristic variant *AC-HH2* was obtained by running SMAC with the following setup: no *cap-time* (each run was executed until finished), wall-clock time of 10 days, and a ratio of 70%-30% for training and testing instances, respectively. Results for this variant and the previous best ones are shown in Table II. Results show that the new AC-HH2 variant outperforms both ESE-HH and AC-HH in terms of average gap. Particularly, *wallclock-time* influences the effectiveness of the result configuration even for instances in which *cap-time* do no represent a comparability issue, i.e, the smaller instances whose tests were executed in less than 480 seconds. Changing from 2 to 10 days generates an impact that is pretty clear in the average gap over all instances.

C. Configuring by Group Instances

An additional issue was considered in the performance of the SMAC provided configuration. Something particular in results of Table I were the poor results obtained in the most complex instances regarding the more easy ones. Even if this may be influenced by the *cap-time*, there was highly likely that several parameters of the configured models would have a heterogeneous performance regarding instance features. In other words, there may be extremely complex to obtain a single configuration that would adequately work over all problem instances.

Considering the above observations, a new configuration process was performed over the subset of instances with 5 constraints only, using exactly the same configuration of AC-HH2, except for the *wall-clock* time, which was only 1 day. The results of this new configuration, denoted by *AC-HH3*, are shown in Table III along with the previous hyper-heuristic variants. It is interesting that the new AC-HH3 outperforms all the others, even if ESE-HH outperforms AC-HH and AC-HH2. Although more test on the remaining instances are advised to

TABLE II
RESULTS OBTAINED FOR THE DIFFERENT COMBINATIONS OF HYPER-HEURISTICS MODELS, ON EACH GROUP OF INSTANCE FEATURES FOR THE MKP. A NEW SMAC CONFIGURED HYPER-HEURISTIC IS ADDED TO THE COMPARISON (AC-HH2), WHICH OUTPERFORM THE PREVIOUS ONES. THE RESULTS ARE SHOWN AS THE AVERAGE VALUE (OVER 20 REPETITIONS) OF THE GAP (THE LOWER, THE BETTER).

			Model		
cons	**vars**	**tigh**	ESE-HH	AC-HH	AC-HH2
5	100	0.25	0.01172	0.01236	0.01253
		0.5	0.00616	0.00635	0.00612
		0.75	0.00418	0.00432	0.00431
	250	0.25	0.01155	0.01228	0.01087
		0.5	0.00689	0.00687	0.00600
		0.75	0.00379	0.00390	0.00339
	500	0.25	0.01810	0.01712	0.01492
		0.5	0.01193	0.01062	0.00946
		0.75	0.00644	0.00613	0.00537
10	100	0.25	0.02033	0.02205	0.02165
		0.5	0.01055	0.01112	0.01121
		0.75	0.00604	0.00614	0.00612
	250	0.25	0.01603	0.01727	0.01567
		0.5	0.00869	0.00901	0.00831
		0.75	0.00479	0.00510	0.00452
	500	0.25	0.02189	0.02160	0.01906
		0.5	0.01196	0.01168	0.01031
		0.75	0.00620	0.00669	0.00574
30	100	0.25	0.03568	0.03784	0.03695
		0.5	0.01618	0.01728	0.01708
		0.75	0.00951	0.01036	0.01001
	250	0.25	0.02343	0.02595	0.02484
		0.5	0.01127	0.01201	0.01141
		0.75	0.00586	0.00643	0.00618
	500	0.25	0.02319	0.02441	0.02281
		0.5	0.01078	0.01140	0.01072
		0.75	0.00588	0.00650	0.00609
		AVG	0.01219	0.01269	0.01191

TABLE III
RESULTS OBTAINED FOR THE DIFFERENT COMBINATIONS OF HYPER-HEURISTICS MODELS, ON GROUPS OF INSTANCE FEATURES WITH 5 CONSTRAINTS FOR THE MKP. A NEW SMAC CONFIGURED HYPER-HEURISTIC IS ADDED TO THE COMPARISON (AC-HH3), WHICH WAS CONFIGURED EXCLUSIVELY WITH THIS GROUP OF INSTANCES AND WHICH OUTPERFORM THE PREVIOUS ONES. THE RESULTS ARE SHOWN AS THE AVERAGE VALUE (OVER 20 REPETITIONS) OF THE GAP (THE LOWER, THE BETTER).

			Model			
cons	**vars**	**tigh**	ESE-HH	AC-HH	AC-HH2	AC-HH3
5	100	0.25	0.01172	0.01236	0.01253	0.01171
		0.5	0.00616	0.00635	0.00612	0.00603
		0.75	0.00418	0.00432	0.00431	0.00415
		AVG	0.00735	0.00768	0.00765	0.00730

generalize the following idea, these results provide insights that a *partial configuration* can effectively improve the results for a specific group of instances, even if a low wall-clock time is used. Moreover, it is possible that running partial configurations for longer days can allow to find the most effective configurations for the problem.

VI. CONCLUSIONS AND FUTURE WORK

In this work, several hyper-heuristic solvers were evaluated in terms of performance and when applied to the MKP problem. Particularly, specific variants of such solvers were generated by applying an automatic parameter configuration process. For this, SMAC was used to find the most adequate parameter values, including the enabling or disabling of algorithm components. After several evaluations of the tool configuration, results across all instances of the problem were effectively improved regarding the default values selected by the solver designer. This allowed to validate that SMAC is a pretty convenient approach to automatic configuration, an important issue regarding the evaluation of parameterized algorithms under empirical conditions.

Several issues regarding the comparisons performed in this work can be detailed to gather new relevant information of SMAC usage. For example, separate studies can be performed to evaluate the impact of using different cap-times, wall-clock times and train and test instances. Additionally, it should be interesting to provide a better generalization regarding the efficiency of partial configurations, which can represent an interesting approach in the process of automated configuration. Additionally, by seizing the cross-domain nature of hyper-heuristics, it will be interesting to evaluate the performance of these configurations across heterogeneous problem domains.

The approach of including automated configuration mechanism alongwith practical optimization techniques such as hyper-heuristics could allow the implementation of strong decision-support systems in practical environments. Rather than focusing on particularities of the problem solved through the manual adjusting of high-level parameters, the algorithm designer could focus more on domain-related parameters directly related to scenario adaption, e.g., the definition of weight values on the objective function. This would support more flexible and useful systems that can reach a more broader range of user information requirements.

To the knowledge of the authors, this is the first approach in using SMAC for configuring hyper-heuristic solvers. With this work, the authors expect to effectively incentive the use of automated parameter configuration tools in their high-level solvers, with the validation benefits that this involve to their research.

REFERENCES

[1] F. Hutter, H. H. Hoos, and T. Stützle, "Automatic Algorithm Configuration Based on Local Search," in *AAAI*. AAAI Press, 2007, pp. 1152–1157. [Online]. Available: http://www.aaai.org/Library/AAAI/2007/aaai07-183.php

[2] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An Automatic Algorithm Configuration Framework," *J. Artif. Intell. Res. (JAIR)*, vol. 36, pp. 267–306, 2009. [Online]. Available: http://dx.doi.org/10.1613/jair.2861

[3] P. Cowling, G. Kendall, and E. Soubeiga, "A Hyperheuristic Approach to Scheduling a Sales Summit," *Lecture Notes in Computer Science*, vol. 2079, pp. 176—-190, 2001. [Online]. Available: http://link.springer-ny.com/link/service/series/0558/papers/2079/20790176.pdf

[4] G. Kendall, P. Cowling, and E. Soubeiga, "Choice function and random hyperheuristics," in *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*. Citeseer, 2002, pp. 667–671.

[5] P. Cowling, G. Kendall, and E. Soubeiga, "Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation," in *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim*, ser. LNCS, S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. Raidl, Eds., vol. 2279. Kinsale, Ireland: Springer-Verlag, 2002, pp. 1–10.

[6] S. Luke, *Essentials of Metaheuristics*. Lulu, 2009.

[7] K. Sörensen and F. W. Glover, "Metaheuristics," in *Encyclopedia of Operations Research and Management Science*. Springer, 2013, pp. 960–970.

[8] K. Sörensen, "Metaheuristics—the metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.

[9] F. Hutter, H. Hoos, and K. Leyton-Brown, "Sequential Model-Based Optimization for General Algorithm Configuration (extended version)," University of British Columbia, Department of Computer Science, Tech. Rep. TR-2010-10, 2010.

[10] A. Fréville, "The multidimensional 0-1 knapsack problem: An overview," *European Journal of Operational Research*, vol. 155, no. 1, pp. 1–21, 2004. [Online]. Available: http://dx.doi.org/10.1016/S0377-2217(03)00274-1

[11] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-Heuristics: An Emerging Direction in Modern Search Technology," in *Handbook of Metaheuristics*, ser. International Series in Operations Research & Management Science, 2003, ch. 16, pp. 457–474. [Online]. Available: http://dx.doi.org/10.1007/0-306-48056-5_16

[12] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, "A Classification of Hyper-heuristics Approaches," in *Handbook of Metaheuristics*, 2nd ed., ser. International Series in Operations Research & Management Science, M. Gendreau and J.-Y. Potvin, Eds. Springer, 2010, vol. 57, ch. 15, pp. 449–468.

[13] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intell. Data Anal*, vol. 12, no. 1, pp. 3–23, 2008.

[14] M. Ayob and G. Kendall, "A Monte Carlo Hyper-Heuristic To Optimise Component Placement Sequencing For Multi Head Placement Machine," in *Placement machine, INTECH03 Thailand*, 2003, pp. 132–141.

[15] P. Cowling, G. Kendall, and E. Soubeiga, "A parameter-free hyper-heuristic for scheduling a sales summit," in *Proceedings of the 4th metaheuristic international conference*, vol. 1101. Citeseer, 2001, pp. 127–131.

[16] E. Burke and E. Soubeiga, "Scheduling Nurses Using a Tabu-search Hyperheuristic," in *In Proceeding of the 1st Multidisciplinary International Scheduling Conference (MISTA 2003*, 2003, pp. 197–218.

[17] G. Kendall and M. Mohamad, "Channel Assignment in Cellular Communication Using a Great Deluge Hyper-Heuristic," in *in the Proceedings of the 2004 IEEE International Conference on Network (ICON2004*, 2004, pp. 769–773.

[18] J. Beasley and P. C. Chu, "A genetic algorithm for the multidimensional knapsack problem," *Journal of Heuristics*, vol. 4, pp. 63–86, 1998.

[19] X. Kong, L. Gao, H. Ouyang, and S. Li, "Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm," *Computers & OR*, vol. 63, pp. 7–22, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.cor.2015.04.018

[20] Y. Yoon and Y.-H. Kim, "A Memetic Lagrangian Heuristic for the 0-1 Multidimensional Knapsack Problem," *Discrete Dynamics in Nature and Society*, 2013. [Online]. Available: http://dx.doi.org/10.1155/2013/474852;

[21] Y. Akçay, H. Li, and S. H. Xu, "Greedy algorithm for the general multidimensional knapsack problem," *Annals OR*, vol. 150, no. 1, pp. 17–29, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10479-006-0150-4

[22] J. H. Drake, E. Özcan, and E. K. Burke, "Modified Choice Function Heuristic Selection for the Multidimensional Knapsack Problem," in *ICGEC*, ser. Advances in Intelligent Systems and Computing, H. Sun, C.-Y. Yang, C.-W. Lin, J.-S. Pan, V. Snásel, and A. Abraham, Eds., vol. 329. Springer, 2014, pp. 225–234. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-12286-1

[23] ——, "A case study of controlling crossover in a selection hyper-heuristic framework using the multidimensional knapsack problem," *Evolutionary computation*, 2015.

[24] J. H. Drake, M. Hyde, K. Ibrahim, and E. Ozcan, "A genetic programming hyper-heuristic for the multidimensional knapsack problem," *Kybernetes*, vol. 43, no. 9/10, pp. 1500–1511, 2014. [Online]. Available: http://www.emeraldinsight.com/doi/full/10.1108/K-09-2013-0201

[25] E. Urra, C. Cubillos, and D. Cabrera-Paniagua, "A Hyperheuristic for the Dial-a-Ride Problem with Time Windows," *Mathematical Problems in Engineering*, vol. 2015, 2015.