An Intelligent Hyper-heuristic Framework for CHeSC 2011

M. Mısır[1,2], K. Verbeeck[1,2], P. De Causmaecker[2], and G. Vanden Berghe[1,2]

[1] CODeS, KAHO Sint-Lieven
{mustafa.misir,katja.verbeeck,greet.vandenberghe}@kahosl.be
[2] CODeS, Department of Computer Science, K.U.Leuven Campus Kortrijk
patrick.decausmaecker@kuleuven-kortrijk.be

Abstract. The present study proposes a new selection hyper-heuristic providing several adaptive features to cope with the requirements of managing different heuristic sets. The approach suggested provides an intelligent way of selecting heuristics, determines effective heuristic pairs and adapts the parameters of certain heuristics online. In addition, an adaptive list-based threshold accepting mechanism has been developed. It enables deciding whether to accept or not the solutions generated by the selected heuristics. The resulting approach won the first Cross Domain Heuristic Search Challenge against 19 high-level algorithms. The detailed empirical results concerning the behaviour of the hyper-heuristic and its sub-mechanisms will be presented at the conference.

1 Introduction

Selection hyper-heuristics are high-level search and optimisation strategies operating on a set of low-level heuristics in a problem-independent manner [1, 2]. In this study, we developed an intelligent selection hyper-heuristic to deal with different heuristic sets for six problem domains provided by HyFlex [3]. The development phase consists of determining the generality requirements, designing effective components with online adaptation skills for these requirements, and combining them using certain decision mechanisms. The empirical results showed that the proposed approach is capable of delivering high performance over the tested problems.

2 Methodology

2.1 Adaptive Dynamic Heuristic Set Strategy

The adaptive dynamic heuristic set (ADHS) strategy [4, 5] is responsible for monitoring the performance of each heuristic to determine elite heuristic subsets for consecutive iteration blocks, each referring to one particular phase. The underlying motivation is to specify the best performing heuristics that will be used during a number of phases to make the heuristic selection process easier. A performance metric (pm_i) based on simple quality indicators such as improvement

capability and speed, is used to decide upon exclusion of a heuristic. Equation 1 illustrates how the performance of heuristic i is measured. In this equation, $C_{p,best}(i)$ is the number of new best solutions explored. $f_{imp}(i)$ and $f_{wrs}(i)$ show the total improvement and worsening during the whole run. $f_{p,imp}(i)$ and $f_{p,wrs}(i)$ indicate the improvement and worsening provided during the current phase. t_{remain} is the remaining execution time. $t_{spent}(i)$ refers to the total time spent until that moment and $t_{p,spent}(i)$ demonstrates the execution time spent during the current phase. For each contributing performance element, a weight w_i is utilised. The values of these weights are set in a decreasing manner. The weights are sufficiently different to manage them in order of importance.

$$
\begin{aligned}
pm_i = w_1 \Big[\Big(C_{p,best}(i) + 1 \Big)^2 \Big(t_{remain} / t_{p,spent}(i) \Big) \Big] \times b + \\
w_2 \Big(f_{p,imp}(i) / t_{p,spent}(i) \Big) - w_3 \Big(f_{p,wrs}(i) / t_{p,spent}(i) \Big) + \\
w_4 \Big(f_{imp}(i) / t_{spent}(i) \Big) - w_5 \Big(f_{wrs}(i) / t_{spent}(i) \Big) \qquad (1) \\
b = \begin{cases} 1, & \sum_{i=0}^{n} C_{p,best}(i) > 0 \\ 0, & otw. \end{cases}
\end{aligned}
$$

The corresponding p_i values are ranked and a quality index ($QI \in [1, n]$) value is determined for each heuristic based on this ranking as a normalisation of the p_i values. The best performing heuristic gets the highest QI that is the number of heuristics (n) currently available. The QI values of the remainder of the heuristics decrease by 1 for each ranking level and the heuristic with the lowest p_i value has $QI = 1$. The heuristics with a QI less than the average of QIs are excluded, which means that it will not be called upon for a number of phases. These excluded heuristics have also $QI = 1$. The length of exclusion is called tabu duration d and it is set to $\sqrt{2n}$. If a heuristic is consecutively excluded, its tabu duration is incremented by 1. Alternatively, if a heuristic is not excluded after performing a phase, its tabu duration is set back to the initial value. This incrementation continues until the corresponding tabu duration reaches its upper bound, which is set to $2\sqrt{2n}$. Whenever the tabu duration is equal to its upper bound, ADHS permanently excludes this heuristic.

The phase length (pl) is set to ($d \times ph_{factor}$) iterations. ph_{factor} is a predetermined constant and it is set to 500. For instance, if the number of heuristics in the heuristic set is 10, then the tabu duration is set as $d = 4$ and pl is 2000 iterations. Whenever the heuristic subset is updated, pl is adjusted with respect to the average time required for performing a move by a non-tabu heuristic. This adjustment was performed based on the number of phases requested ($ph_{requested} = 100$) which is a predefined value, as illustrated in Equation 2. $C_{moves}(i)$ shows the number of times heuristic i is called and t_{total} indicates the total execution time. The calculated value is constantly checked to keep it within its bounds, $pl \in [d \times 50, d \times ph_{factor}]$.

$$pl = \Big(t_{total}/ph_{requested}\Big) / \sum_{i=0}^{n} \Big(t_{spent}(i)/C_{moves}(i)\Big).isTabu(i) \quad (2)$$

Extreme heuristic exclusion : Some of the heuristics which did not find new best solutions during a phase are additionally excluded based on Equation 3 at the end of each phase. The idea behind this extra exclusion procedure is to fasten the search process by eliminating slow heuristics compared to the speed of the other heuristics in the heuristic set. The standard deviation (σ) and the average (ϖ) of the $exc(i)$ values together with the number of new best solutions (nb) found by the heuristics in the heuristic set are used for this additional exclusion as shown in Equation 4.

$$exc(i) = \Big(t_{spent}(i)/C_{moves}(i)\Big) / \Big(t_{spent}(fastest)/C_{moves}(fastest)\Big) \quad (3)$$

$$\sigma > 2.0 \ ; \ exc(i) > 2\varpi \ ; \ nb > 1 \quad (4)$$

Heuristic selection : In order to choose a heuristic from the heuristic subset, a probability vector is maintained. The selection probabilities of the heuristics are the normalisation of the calculated values based on Equation 5.

$$pr_i = \big((C_{best}(i) + 1)/t_{spent}\big)^{(1+3tf^3)} \quad (5)$$

$$tf = (t_{total} - t_{elapsed})/t_{total}$$

2.2 Relay Hybridisation

The hyper-heuristic also investigates a simple relay hybridisation approach to determine effective pairs of heuristics that are applied consecutively. The details of this approach are presented in Algorithm 1. C_{phase} denotes the number of iterations that have been executed during the current phase. $C_{best,s}$ is a counter regarding the number of new best solutions found by the single heuristic selection method. $C_{best,r}$ is another counter for the number of new best solutions found by the relay hybridisation. p is a random variable to decide upon using relay hybridisation. p' is another random variable for choosing the second heuristic. $list_i$ indicates the list of heuristics to be applied after heuristic i. The size of the list is set to 10 for each heuristic. The choice of the first heuristic is made by a learning automaton (LA) that keeps a probability list with the selection probabilities of the first heuristics [6]. A *linear reward-inaction* update scheme is used for updating the probabilities as indicated in Equation 6 and 7. In these equations, the learning rates are set as $\lambda_1 = 0.5$ and $\lambda_2 = 0$. This update scheme increases the probability of a heuristic that has found new best solutions.

In addition, the tabu approach used for ADHS is applied to disable relay hybridisation if it could not deliver a new best solution after a phase.

Algorithm 1: Relay hybridisation

Input: $list_{size} = 10; \gamma \in (0.02, 50); p, p' \in [0:1]$
$\gamma = (C_{best,s} + 1)/(C_{best,r} + 1)$
if $p \leq (C_{phase}/pl)^{\gamma}$ **then**
  select LLH using a LA and apply to $S \rightarrow S'$
  if $size(list_i) > 0$ *and* $p' <= 0.25$ **then**
    select a LLH from $list_i$ and apply to $S' \rightarrow S''$
  else
    select a LLH and apply to $S' \rightarrow S''$
  end
end

$$p_i(t+1) = p_i(t) + \lambda_1 \beta(t)(1 - p_i(t)) - \lambda_2(1 - \beta(t))p_i(t) \quad \text{if } a_i \text{ is the action taken at time step } t \tag{6}$$

$$p_j(t+1) = p_j(t) - \lambda_1 \beta(t)p_j(t) + \lambda_2(1 - \beta(t))[(r-1)^{-1} - p_j(t)] \quad \text{if } a_j \neq a_i \tag{7}$$

2.3 Heuristic Parameter Adaptation

Certain heuristics have a parameter called "*intensity of mutation*" representing the perturbation level. The other heuristics concentrating on improvement only have a parameter called "*depth of search*" related to the number of steps to be applied. A reward-penalty strategy is used to dynamically adapt these parameters.

2.4 Adaptive Iteration Limited List-based Threshold Accepting

Adaptive iteration limited list-based threshold accepting (AILLA) is a move acceptance mechanism providing an adaptive diversification strategy in connection with the quality of the explored new best solutions earlier [4, 7, 5]. Its details are presented in Algorithm 2.

The iteration limit (k) is updated as shown in Equation 8. For the list length (l), the update rule presented in Equation 6 is utilised ($l_{base} = 5$, $l_{initial} = 10$).

$$k = \begin{cases} ((l-1) \times k + iter_{elapsed})/l, & \text{if } cw = 0 \\ ((l-1) \times k + \sum_{i=0}^{cw} k \times 0.5^i \times tf)/l, & \text{otherwise} \end{cases} \tag{8}$$

$$cw = iter_{elapsed}/k$$

$$l = l_{base} + (l_{initial} - l_{base} + 1)tf^3 \tag{9}$$

Algorithm 2: AILLA move acceptance

```
   Input: i = 1, K ≥ k ≥ 0, l > 0
   for i=0 to l-1 do best_list(i) = f(S_initial)
1  if adapt_iterations ≥ K then
2  |  if i < l − 1 then
3  |  |  i + +
   |  end
   end
4  if f(S′) < f(S) then
5  |  S ← S′
6  |  w_iterations = 0
7  |  if f(S′) < f(S_b) then
8  |  |  i = 1
9  |  |  S_b ← S′
10 |  |  w_iterations = adapt_iterations = 0
11 |  |  best_list.remove(last)
12 |  |  best_list.add(0, f(S_b))
   |  end
13 else if f(S′) = f(S) then
14 |  S ← S′
15 else
16 |  w_iterations + +
17 |  adapt_iterations + +
18 |  if w_iterations ≥ k and f(S′) ≤ best_list(i) then
19 |  |  S ← S′ and w_iterations = 0
   |  end
   end
```

Re-initialisation : The threshold level ($best_{list}(i)$) starts from the lowest value and increases to the value placed in the l th location of the list. Each time the threshold level reaches value l, a new initial solution is randomly generated to find new best solutions in a faster way. Re-initialisation is disabled depending on the remaining execution time, its cost and the possibility of finding a new best solution afterwards.

3 Results and Conclusion

This study is about designing an intelligent hyper-heuristic to provide high quality performance across different optimisation problems. The hyper-heuristic presented here was submitted to the first international Cross-domain Heuristic Search Challenge (CHeSC 2011) to show its generality and robustness across multiple problem domains. It ended up as the competition winner out of 20 submissions. The performance of the competing algorithms were compared for five instances from six problem domains, i.e. max SAT, 1D bin packing, permutation flowshop scheduling, personnel scheduling, travelling salesman, vehicle routing. The last two domains were added to the problem set as hidden domains. The ranking and scores[1] of the corresponding algorithms are shown in Table 1. A detailed experimental analysis is available in [8].

[1] `http://www.asap.cs.nott.ac.uk/chesc2011/results.html`

Table 1. CHeSC 2011 competition ranking and scores

Algorithm	**Overall Score**
ADAPHH (**Our method**)	181
VNS-TW	134
ML	131.5
PHUNTER	93.25
EPH	89.75
HAHA	75.75
NAHH	75
ISEA	71
KSATS-HH	66.5
HAEA	53.5
ACO-HH	39
GenHive	36.5
DynILS	27
SA-ILS	24.25
XCJ	22.5
AVEG-Nep	21
GISS	16.75
SelfSearch	7
MCHH-S	4.75
Ant-Q	0

References

1. Ozcan, E., Misir, M., Ochoa, G., Burke, E.: A reinforcement learning - great-deluge hyper-heuristic for examination timetabling. International Journal of Applied Metaheuristic Computing **1**(1) (2010) 39–59
2. Burke, E., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society (to appear)
3. Burke, E., Curtois, T., Hyde, M., Ochoa, G., Vazquez-Rodriguez, J.A.: HyFlex: A benchmark framework for cross-domain heuristic search. ArXiv e-prints, arXiv:1107.5462 (2011)
4. Misir, M., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G.: A new hyper-heuristic implementation in HyFlex: a study on generality. In Fowler, J., Kendall, G., McCollum, B., eds.: the 5th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'11), Phoenix/Arizona, USA (2011) 374–393
5. Misir, M., Smet, P., Verbeeck, K., Vanden Berghe, G.: Security personnel routing and rostering: a hyper-heuristic approach. In Gunalay, Y., Kadipasaoglu, S., eds.: Proceedings of the 3rd International Conference on Applied Operational Research (ICAOR'11). Volume 3 of LNMS., Istanbul, Turkey (2011) 193–205
6. Misir, M., Wauters, T., Verbeeck, K., Vanden Berghe, G.: A Hyper-heuristic with Learning Automata for the Traveling Tournament Problem. In: Metaheuristics: Intelligent Decision Making, the 8th Metaheuristics International Conference - Post Conference Volume. Springer (to appear)
7. Misir, M., Vancroonenburg, W., Vanden Berghe, G.: A selection hyper-heuristic for scheduling deliveries of ready-mixed concrete. In: Proceedings of the 9th Metaheuristic International Conference (MIC'11), Udine, Italy (2011)
8. Misir, M., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G.: Design and analysis of an evolutionary selection hyper-heuristic. Tech. report, KAHO Sint-Lieven (2011)