# An adaptive hyper-heuristic for CHeSC 2011

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

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

## 1 Introduction

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 was developed. It enables deciding whether to accept the solutions generated by the selected heuristics or not. The details of these mechanisms are presented in the following sections.

## 2 Maintaining an Adaptive Dynamic Heuristic Set

Maintaining an adaptive dynamic heuristic set (ADHS) is a method assessing the performance of each heuristic at the end of a number of iterations, i.e. a phase, to keep the best performing heuristics in the set whilst excluding the others. A performance metric based on simple quality indicators such as improvement capability and speed, is used to decide upon exclusion of a heuristic. The details of this metric regarding heuristic $i$ is shown in Equation 1. In this equation, $C_{p,best}(i)$ denotes the number of new best solutions found. $f_{imp}(i)$ and $f_{wrs}(i)$ show the total improvement and worsening provided. $f_{p,imp}(i)$ and $f_{p,wrs}(i)$ refer to the same measurement but only in a phase. $t_{remain}$ refers to the remaining time to finish the whole search process. $t_{spent}(i)$ and $t_{p,spent}(i)$ are the time spent by heuristic $i$ until that moment and the same measurement during a phase respectively. 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.

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

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. The heuristics with a $QI$ less than the average of $QI$s are excluded, which means that it will not be called for a number of phases. This number 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 for a proper $pl$ and it is set to 500. 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. As a reference point, the number of phases requested ($ph_{requested} = 100$) during the whole run is used to determine a possible phase duration. $pl \in [d \times ph_{base}, d \times ph_{factor}]$ is calculated based on Equation 2. $C_{moves}(i)$ shows the number of times heuristic $i$ is called. The calculated value is constantly checked to keep it within its bounds.

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

**Extreme heuristic exclusion** : At the end of each phase, some of the heuristics which did not find new best solutions during the phase are additionally excluded based on Equation 3. The standard deviation and the average of the $exc(i)$ values are used to determine the heuristics that should be excluded.

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

In order to select a heuristic from the heuristic subset, a probability vector is maintained. The selection probabilities of the heuristics are calculated as shown in Equation 4.

$$\begin{aligned} pr_i &= \big((C_{best}(i) + 1)/t_{spent}\big)^{(1+3tf^3)} \\ tf &= (t_{exec} - t_{elapsed})/t_{exec} \end{aligned} \tag{4}$$

## 3 Relay Hybridisation

The hyper-heuristic also investigates a simple relay hybridisation approach to determine effective pairs of heuristics that are applied consecutively. For that purpose, a heuristic list of size 10 is maintained for each heuristic. The choice of the first heuristic is made by a learning automaton that keeps a probability list

with the selection probabilities of the first heuristics [1]. A *linear reward-inaction* update scheme is used for updating the probabilities. This scheme increases the probability of a heuristic that found new best solutions. The details are presented in Algorithm 1. In the pseudocode, $C_{phase}$ is a counter showing the number of iterations passed during the current phase. $C_{best,s}$ and $C_{best,r}$ refer to the number of new best solutions found by the regular selection mechanism and the relay hybridisation respectively. $p$ is a random variable to decide upon using relay hybridisation. $p'$ is another random variable for choosing the second heuristic.

**Algorithm 1:** Relay hybridisation

```
   Input: list_size = 10; γ ∈ (0.02, 50); p, p' ∈ [0 : 1]
1  γ = (C_best,s + 1)/(C_best,r + 1)
2  if p ≤ (C_phase/pl)^γ then
3  |   select an LLH using a learning automaton and apply it to S → S'
4  |   if size(list_i) > 0 and p' <= 0.25 then
5  |   |   select an LLH from list_i and apply to S' → S''
6  |   else
7  |   |   select an LLH and apply to S' → S''
   |   end
   end
```

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.

### 3.1 Heuristic Parameter Adaptation

Certain heuristics have a parameter called "*intensity of mutation*" denoting the impact of the perturbation. The other heuristics concentrating on improvement only have a parameter called "*depth of search*" referring to the number of consecutive steps to check for improvement. A reward-penalty strategy is employed to dynamically adapt these parameters.

## 4 Move Acceptance

### 4.1 Adaptive Iteration Limited List-based Threshold Accepting

Adaptive iteration limited list-based threshold accepting (AILLA) is a threshold accepting mechanism determining the threshold level in a dynamic manner using the fitness values of the previously found new best solutions. The details of this mechanism is presented in Algorithm 1.

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

**Algorithm 2:** AILLA move acceptance

**Input**: $i = 1, K \geq k \geq 0, l > 0$
**for** *i=0* **to** *l-1* **do** $best_{list}(i) = f(S_{initial})$
**if** $adapt\_iterations \geq K$ **then**
  **if** $i < l - 1$ **then**
    $i++$
  **end**
**end**
**if** $f(S') < f(S)$ **then**
  $S \leftarrow S'$
  $w\_iterations = 0$
  **if** $f(S') < f(S_b)$ **then**
    $i = 1$
    $S_b \leftarrow S'$
    $w\_iterations = adapt\_iterations = 0$
    $best_{list}.remove(last)$
    $best_{list}.add(0, f(S_b))$
  **end**
**else if** $f(S') = f(S)$ **then**
  $S \leftarrow S'$
**else**
  $w\_iterations++$
  $adapt\_iterations++$
  **if** $w\_iterations \geq k$ *and* $f(S') \leq best_{list}(i)$ **then**
    $S \leftarrow S'$ and $w\_iterations = 0$
  **end**
**end**

$$k = \begin{cases} ((l-1) \times k + iter_{elapsed})/l, & \text{if } cw = 0 \\ ((l-1) \times k + \sum_{i=0}^{cw} k \times 0.5^i \times tf)/l, & \text{otherwise} \end{cases} \quad (5)$$

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

$$l = l_{base} + (l_{initial} - l_{base} + 1)^{tf^3} \quad (6)$$

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. Whenever the threshold level reaches this value $l$, a new initial solution is generated to find new best solutions faster. Depending on the remaining execution time, the cost of reinitialisation and the possibility of finding a new best solution after reinitialisation, reinisialisation is disabled or not.

## References


1. M. Misir, T. Wauters, K. Verbeeck, and G. Vanden Berghe. A new learning hyper-heuristic for the traveling tournament problem. In *Proceedings of the 8th Metaheuristic International Conference (MIC'09)*, Hamburg, Germany, July 13–16 2009.