Published version: A.Salhi, and J.A. Vazquez Rodriguez, "Tailoring Hyper-Heuristics to Specific Instances of a Scheduling Problem Using Affinity and Competence Functions", Journal of Memetic Computing, 6(2), pp. 77-84, 2014,

Tailoring Hyper-Heuristics to Specific Instances of a Scheduling Problem Using Affinity and Competence Functions

Abdellah Salhi[†a]*, and José Antonio Vázquez Rodríguez[+b]

[†] Department of Mathematical Sciences,
The University of Essex,
Colchester CO4 3SQ, UK

[+] Department of Computer Science and Information Technology,
The University of Nottingham,
Wollaton Road, Nottingham NG8 1BB, UK

[a]as@essex.ac.uk, Tel: 00 44 1206 873022, Fax: 00 44 1206 873043
[b]jav@cs.nott.ac.uk, Tel: 00 44 1158 468403, Fax: 00 44 1159 514 254

Abstract

Hyper-heuristics are high level heuristics which coordinate lower level ones to solve a given problem. Low level heuristics, however, are not all as competent/good as each other at solving the given problem and some do not work together as well as others. Hence the idea of measuring how good they are (competence) at solving the problem and how well they work together, (their affinity.) Models of the affinity and competence properties are suggested and evaluated using previous information on the performance of the simple low level heuristics. The resulting model values are used to improve the performance of the hyper-heuristic by tailoring it not only to the specific problem but the specific instance being solved. The test case is a hard combinatorial problem, namely the Hybrid Flow Shop scheduling problem. Numerical results on randomly generated as well as real-world instances are included.

1 Introduction

A special type of heuristics called Hyper-Heuristics (HH) which "coordinate" other heuristics, is now a well established set of tools for the solution of optimisation

*Contact Author: A.Salhi, Dept. of Mathematical Sciences, The University of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK

problems, [1]. HH are high level heuristics that receive as input a set of low level heuristics and a problem. At decision points in the solution process, the HH selects a low level heuristic from a set H and applies it to the current state of the solution. Many such low level heuristics, therefore, may be called during the solution process and the sequence of their appearances is important. HH are particularly suitable for real problems where, "soon-enough", "good-enough" and "cheap-enough" solutions are preferred to optimal but expensive (to obtain) ones [2], and those that are unattainable in the available time. Recent developments in HH can be found in [3, 4].

1.1 Classes of hyper-heuristics

We shall differentiate between two types of hyper-heuristics: improving hyper-heuristics (IHH) and constructive hyper-heuristics (CHH).

1.1.1 Improving hyper-heuristics

In IHH, an initial solution, s^o, is part of the input; the low level heuristics are applied to it in order to improve it. Examples of these are given in [1] and [5]. According to [2], provided a set of improving low level heuristics for the problem in hand is available, the following algorithmic framework (Algorithm 1) is, most of the time, successful.

Algorithm 1 Improving hyper-heuristics

Input: s^o, H
While stopping condition not satisfied **do**
  select $h_i \in H$
  $s^* \leftarrow h_i(s^o)$
  If a certain condition is satisfied **then**
    $s^o \leftarrow s^*$
  End If
End While
Return s^*

Different mechanisms to select low level heuristics from H, such as the choice function, [1], tabu search, [5], GA, [6] and other meta-heuristics, [7], have been proposed. In the same way, different conditions for the acceptance of new solutions have been proposed. These include the acceptance of all improving solutions, [1], simulated annealing, and other similar stochastic based settings, [8].

1.1.2 Constructive hyper-heuristics

In CHH the low level heuristics are used to build solutions from scratch. Each low level heuristic $h_i \in H$ is constructive, i.e. it can be applied on its own to build a solution. There are two types of CHH. In the first, [9, 10], a set of good solutions for different instances of a problem are used to train a machine learning system (case based reasoning). This system selects and records relevant instances with their corresponding "good" combination of low level heuristics. When a new instance is given, the "closest" instance in memory is retrieved and its corresponding heuristics applied to the new instance.

The second type of CHH uses an on-the-fly learning process. These methods use meta-heuristics such as GA, [11, 12, 13], and Tabu Search, [14], to the sequence $\mathcal{H} = \{h^1, \ldots, h^t\}$, where $h^i \in H$, of low level heuristics to be applied in that order to build a solution. They test several such $\mathcal{H}$ sequences within an iterative improving process until they reach a satisfactory solution or satisfy a stopping criterion. In the remaining of this paper, the focus is on CHH.

1.2 Motivation

The limitations of the first type of CHH are that (i) when facing a new problem it is not easy to decide which is the adequate case in memory to retrieve, and (ii) their effectiveness is limited since they do not use information on the specific instance being solved. But, they do have a strength which is that, after training, they are efficient at providing "acceptable" solutions. The second type of CHH is easier to implement and has been relatively more successful, [6], [12]. However, it has been observed that its performance can be improved by using information regarding the effectiveness of the low level heuristics on their own and working with others, [15]. In this paper two functions are suggested to capture such information: the "competence" function, that measures the individual performance of the low level heuristics and the "affinity" function which measures how good pairs of heuristics are at working together. The main thrust of this paper is to combine in a single method the strength of the two kinds of CHH. The proposed method uses GA to search for a combination of low level heuristics aided by the information provided by the affinity and competence functions. In this way, the overall CHH is effectively tailored to the problem instance being solved.

1.3 Organisation of the paper

Section 2 suggests a GA-based CHH (GA-CHH) and presents the affinity and competence functions. The case study, which is the Hybrid Flow Shop (HFS) scheduling problem, is described and briefly reviewed in Section 3. The same

section explains how to adapt the proposed GA-CHH to solve it. In Section 4 two variants of GA-CHH are used to solve a large set of randomly generated HFS instances. The first variant is as described in Subsection 2.1, the second adds to the first one information obtained with the competence and the affinity functions. The two variants are then tested on a set of real-world HFS problems along with a number of well established algorithms namely h_1GA, RKGA [16], GA$_H$ [17], and EDD-W [15]; all will be briefly presented later. Section 5 concludes the paper and points out issues which merit further work.

2 Methodology

2.1 A GA-based constructive hyper-heuristic

A GA-based CHH similar to those found in [12] and [13] is suggested here. Let $H = \{h_1, \ldots, h_N\}$ be the set of low level heuristics. An individual from the population of the GA-CHH is a combination of low level heuristics in the form $\mathcal{H} = \{h^1, \ldots, h^t\}$, where $h^i \in H$. Note that the elements in $\mathcal{H}$ may be repeated. In order to translate a given $\mathcal{H}$ into a solution to the problem in hand, the low level heuristics are invoked in the order $h^1, h^2, \ldots, h^t$ they appear in $\mathcal{H}$, as in Algorithm 2 below.

Algorithm 2 Individual evaluation of GA-CHH

input: $\mathcal{H}$
For $i = 1, \ldots, t$ **do**
| apply $h^i \in \mathcal{H}$ to the current solution state;
End for
End

The genetic operators and selection mechanism used are as in many other GA applications; a two point crossover, random generation of a given gene as the mutation operator, and tournament selection as the selection mechanism, see [18] and [19] for details. The encoding of individuals and GA parameters must be adapted to the specific application in hand. Subsection 3.1 provides the necessary details for a complex scheduling problem such as HFS.

2.2 The competence and the affinity functions

Knowing how effective are low level heuristics when working individually and combined with others may prove a useful approach to problem solving in general. The

way this information is captured here is through the *competence* and *affinity* functions. Intuitively, if there are heuristics that contribute more than others to the solution of a problem, they are expected to be found more frequently than others in good solutions. In the same way, if there are pairs of heuristics with strong affinity, it is expected to find them together, i.e. they are invoked one after the other, or at a relatively close distance from each other in the sequence $\mathcal{H}$. These ideas are explored and formalised here.

Given a set of good combinations of heuristics Θ^*, the competence, $C(h_A \in H)$, of heuristic h_A is defined as:

$$C(h_A \in H) = \sum_{\mathcal{H} \in \Theta^*} \sum_{i=1}^{t} count(h_A, h^i \in \mathcal{H}), \tag{1}$$

where

$$count(h_A, h^i) = \begin{cases} 1 & \text{if } h^i = h_A \\ 0 & \text{otherwise} \end{cases},$$

t is the number of heuristics in the solution representation. Expression (1) computes the frequency with which $h_A \in H$ is found in the set of good solutions Θ^*.

Let the affinity of two heuristics h_A and h_B, $Aff(h_A, h_B)$, represent the frequency of h_A and h_B being in the same solution weighted in proportion to how close they are:

$$Aff(h_A, h_B) = \sum_{\mathcal{H} \in \Theta^*} \sum_{i=1}^{t-1} \sum_{j=i+1}^{t} count\_2(h_A, h_B, h^i \in \mathcal{H}, h^j \in \mathcal{H}), \tag{2}$$

where

$$count\_2(h_A, h_B, h^i, h^j) = \begin{cases} \frac{1}{j-i} & \text{if } h^i = h_A \text{ and } h^j = h_B \\ 0 & \text{otherwise} \end{cases}.$$

Note that given a combination of heuristics $\mathcal{H} = \{h^1, \ldots, h^t\}$, we define the distance between two of its elements h^i and h^j, $j > i$, simply as $j - i$. In expression (2), the maximum affinity is achieved when $\frac{1}{j-i} = 1$, which happens when h_A and h_B are adjacent (called one after the other). The value of $\frac{1}{j-i}$ decreases as the two heuristics are found far apart; it is null when the heuristics do not appear in the same solution at all. Let A_{Aff} be the *affinity matrix*, i.e. the affinity measures between all heuristics in H:

$$A_{Aff} = \begin{pmatrix} Aff(h_1, h_1) & \ldots & Aff(h_1, h_{|H|}) \\ \vdots & \ddots & \vdots \\ Aff(h_{|H|}, h_1) & \ldots & Aff(h_{|H|}, h_{|H|}) \end{pmatrix}.$$

The affinity of a heuristic with itself, i.e. $Aff(h_i, h_i)$, measures how profitable it is to make consecutive calls to it.

The competence and affinity functions may be used to sample for combinations of heuristics as follows. Let the probability $P(h)$ of assigning heuristic $h \in H$ to h^1 (i.e. to the first decision) be:

$$P(h) = \frac{C(h)}{\sum_{h_i \in H} C(h_i)}. \tag{3}$$

This means that the choice is biased toward heuristics that are good for the problem. But, if we have already made a choice to fill the previous place in $\mathcal{H}$, (decision block $d^{(i-1)}$, for example), then it makes sense to look for one that has great affinity with it. Therefore, one can also devise a probability based on the affinitiy function as follows. The probability $P(h)$ of assigning heuristic $h \in H$ to h^i, is calculated with the following formula:

$$P(h) = \frac{Aff(h_*, h)}{\sum_{h_i \in H} Aff(h_*, h_i)}, \tag{4}$$

where h_* is the heuristic that was assigned to the previous decision block, $d^{(i-1)}$. Which formula is used is determined by a parameter $0 \leq \alpha \leq 1$; Formula 4 with probability α and Formula 3 with probability $1 - \alpha$. This parameter must be tuned for the specific application depending on whether we have any preference for one function or the other. In the following α takes value 0.5. The procedure for choosing heuristics to build up solutions for HHS is given as Algorithm 3.

3 Case study: the Hybrid Flow Shop scheduling problem

A HFS is a manufacturing environment in which a set of n jobs are to be processed in a series of m stages. There are many variations to this environment, all of which have the following in common:

1. the number of processing stages m is at least 2,
2. each stage k has $m_k \geq 1$ machine in parallel and in at least one of the stages $m_k > 1$,
3. all jobs are processed following the same production flow: stage 1, stage 2,..., stage m. A job can skip any number of stages provided it is processed in at least one of them.

Algorithm 3 Generating new solutions using the information provided by the competence and the affinity functions

input:
$t = n \times m$, where n is the number of jobs, and m the number of stages.
$\alpha = 0.5$.
H, the set of low level heuristics.
$h^1 \leftarrow$select $h \in H$ with probability $\frac{C(h)}{\sum_{h_j \in H} C(h_j)}$.
$h_* \leftarrow h^1$
For $i = 2, \ldots, t$ **do**
  If $(rand() \leq \alpha)$ **then**
    $h^i \leftarrow$select $h \in H$ with probability $\frac{Aff(h_*,h)}{\sum_{h_j \in H} Aff(h_*,h_j)}$
  else
    $h^i \leftarrow$select $h \in H$ with probability $\frac{C(h)}{\sum_{h_j \in H} C(h_j)}$
  End If
  $h_* \leftarrow h^i$
End For
$\mathcal{H} \leftarrow \{h^1, \ldots, h^t\}$
return $\mathcal{H}$

In the case considered here, any machine can process at most one job at a time and any job is processed on at most one machine at a time. Furthermore, every job is processed on at most one machine in any stage. Preemptions are not allowed. The problem is to find a schedule which minimises some objective function.

In most cases, HFS is NP-hard, [20], [21]. It is found in all kinds of real world scenarios such as the electronics, [22], paper, [23], textile, [24], and concrete, [25], industries. Examples are also found in the manufacturing of photographic film, [26], internet services architectures, [27], container handling systems, [28], and others, [29]. Recent solution approaches can be found in [30].

When searching in the set of semi-active schedules, or schedules for which an operation cannot start earlier without changing the order of processing in any one of the machines, [31], then the following formulation is appropriate. Let A^{kl} be a set of operations o_{jk} assigned to machine l in stage k. Let S^{kl} be a sequence of the elements in A^{kl} representing the order in which they are to be processed. Let $S^k = \cup_{l=1}^{m_k} S^{kl}$ where m_k is the number of machines at stage k, and $S = \cup_{k=1}^{m} S^k$. As S^k represents the set of sequences to be followed by the jobs when in stage k, S is a schedule. For it to be feasible, the following must hold.

$$\bigcup_{l=1}^{m_k} A^{kl} = O_k, \forall k \tag{5}$$

$$\bigcap_{l=1}^{m_k} A^{kl} = \emptyset, \forall k \tag{6}$$

Constraints 5 and 6 guarantee that all operations to be processed in stage k, O_k, are assigned for processing strictly once.

Given that S is easily translated into a unique schedule, no difference is assumed between S and the schedule it represents and hereafter S will be referred to as a schedule, S^k as a schedule for stage k, and S^{kl} as a schedule for machine l in stage k.

Let π be a HFS problem instance and Ω^π the set of all schedules that satisfy restrictions 5 and 6 for π. The problem can now be formulated as:

$$\min_{S \in \Omega^\pi} f_i(S)$$

which means, given a problem instance π, find a schedule $S \in \Omega^\pi$ that minimises a cost function $f_i(S)$. $f_i(S)$ is anyone of the following objective functions.

- $f_1 = \max_j C_j$, the makespan or maximum completion time;
- $f_2 = \sum_j C_j$, the sum of completion times;
- $f_3 = \max_j T_j$, where $T_j = \max\{0, C_j - d_j\}$, the maximum tardiness;

- $f_4 = \sum_j \omega_j T_j$, the total weighted tardiness;
- $f_5 = \sum_j \omega_j U_j$, the weighted number of tardy jobs;
- $f_6 = \sum_j \omega_j W_j$, where $W_j = C_j - start_{j1}$,is the total weighted time that job j spends in the system after its start time in stage 1. Note that W_j is different from $F_j = C_j - r_{j1}$ in that the former does not take into consideration the waiting time in the queue previous to stage 1, i.e. $f_6 = F_j - (r_{j1} - start_{j1})$. $start_{j1}$ is the start time of job j in stage 1.
- $f_7 = \max_k \left\{ \max_{time=0}^{C_{\max}} \left\{ \sum_j \omega_j x_{jkt} \right\} \right\}$, where $time \in Z^+$ is the time, $x_{jkt} \in \{1, 0\}$, (it is 1 if $r_{jk} \leq time$ and $v_{jk} \geq time$, it is 0 otherwise); f_7 is the maximum weighted number of jobs waiting for processing at the same time in the same stage;
- $f_8 = \sum_j \omega_j E_j + \sum_j \omega_j T_j$, is the sum of the weighted earliness and tardiness.

Functions f_1 and f_2 are the most widely studied in the literature of HFS. f_1, or maximum completion time, optimises the use of machines in the shop. f_2, or total weighted tardiness, is a good metric of the quality of the service provided. f_3 is the maximum tardiness, f_4 the sum of completion times and f_5 the weighted sum of tardy jobs. f_3 and f_5 are concerned with the quality of the service to the client, while f_4 is concerned with how fast the jobs are completed. f_6 is the sum of weighted waiting times, i.e. it measures the total time that the jobs spend on the shop floor from the moment they start processing in the first stage, until completion. f_7 is the maximum wighted number of jobs that are in a stage waiting for processing at the same time. f_6 and f_7 can be associated with the inventory of products in process. f_8 is the weighted earliness and tardiness. It is relevant in *Just In Time* manufacturing systems where delays to meet demand and delivery are penalised.

The 13 low level heuristics used here are described in Table 1.

3.1 A constructive hyper-heuristic to solve HFS

A GA-CHH as described in Subsection 2.1, was implemented for the HFS problem. We refer to it as the Hyper-Heuristic Scheduler (HHS). In HHS, H is a set of 13 low level heuristics, otherwise known as dispatching rules of the type First In First Out (FIFO), Shortest Processing Time (SPT) and so on, [31], [32]. A solution is a set $\mathcal{H} = \{h^1, \ldots, h^t\}$, where $t = n \times m$. In this application, the first n elements

Table 1: Low level heuristics

Name	Description: **select** $o_{jk} \in O_k$ with
h_1	smallest r_{jk} value; r_{jk} is the release time of o_{jk} operation;
h_2	smallest p_{jk} value; p_{jk} is the processing time of operation o_{jk};
h_3	largest p_{jk} value;
h_4	smallest $(v_{jk} - d_j)$ value; v_{jk} is work remaining for job j in stage k;
h_5	largest $(v_{jk} - d_j)$ value; d_j is the due date of job j;
h_6	smallest v_{jk} value;
h_7	largest v_{jk} value;
h_8	smallest $\omega_j p_{jk}$ value; ω_j is the weight of job j;
h_9	largest $\omega_j p_{jk}$ value;
h_{10}	smallest $\omega_j(v_{jk} - d_j)$ value;
h_{11}	largest $\omega_j(v_{jk} - d_j)$ value;
h_{12}	smallest $\omega_j v_{jk}$ value;
h_{13}	largest $\omega_j v_{jk}$ value;

in $\mathcal{H}$ are used to schedule stage 1, from h^{n+1} to h^{2n} for stage 2, and so on:

$$\overbrace{\underbrace{h^1, \ldots, h^n}_{\text{stage 1,}}, \underbrace{h^{n+1}, \ldots, h^{2n}}_{\text{stage 2,}}, \underbrace{\ldots}_{\text{...,}}, \underbrace{h^{t-n}, \ldots, h^t}_{\text{stage } m}}^{\mathcal{H}}.$$

Every low level heuristic acts as follows: given a set of not yet scheduled operations, O_k, select operation $o \in O_k$, that best satisfies a certain condition. For instance, SPT selects an o that requires the shortest processing time. The operation is assigned to the machine that allows it to be completed in the fastest (completion) time. It is then removed from O_k. To build a full solution the process is repeated until O_k is empty ($n \times m$ times). See Algorithm 4.

After a pre-experimental tuning stage, the following parameter setting was found to be adequate: a crossover probability of 0.9, a mutation probability of 0.1, a tournament selection with 2 participants, a population size of 100 individuals and applying elitism, i.e. the best individual found so far is always kept.

4 Computational experience

Two sets of experiments have been carried out. The first concerns establishing that that HHS_{CA}, the HHS aided with the information provided by the competence and the affinity functions, is superior to the pure HHS. The second establishes that HHS_{CA} is superior to a number of well established algorithms for the problem and this on real-world instances.

Algorithm 4 Procedure called by HHS to evaluate individuals

input: $\mathcal{H} = \{h^1, \ldots, h^t\}$
For $k \leftarrow 1, \ldots, m$ **do** // for every stage
  Let O_k be the set of operations to be scheduled in stage k
  $i \leftarrow 1$
  While $O_k \neq \emptyset$
    $u \leftarrow (k-1) \times n + i$
    Select an operation $o \in O_k$ according to h^u;
    Assign o to the machine in stage k that
      will complete it in the fastest (completion) time;
    $O_k \leftarrow O_k \backslash \{o\}$
    $i \leftarrow i + 1$
  End While
End For

4.1 HHS$_{CA}$ versus HHS on randomly generated HFS problem instances

In order to do this, HHS was run 5 times, each time for a 100 generations, on each of a set of 1024 randomly generated HFS instances, [32]. The best results of the 5 runs were recorded. HHS was run again, but this time, using the information of the competence and the affinity functions as follows: HHS is run once as normal, the last population of this first run is used to calculate the competence and the affinity functions. This information is then used to create the new population for the second run of HHS. Each individual is created with Algorithm 3. In the same way, the competence and affinity functions are calculated after the second run and used to initialise the third one. This process is repeated in the remaining runs.

Details on how the HFS instances were generated can be found in [32]. It is important to say, however, that the test set contains an equal number of instances with $n \in \{20, 40, 60, 80\}$. Eight objective functions, $f_1, \ldots, f_8$, are considered (one at a time). All 1024 instances were considered with each of such functions. The mean result of the best solutions found by HHS and HHS$_{CA}$ for every instance, with the instances grouped according to the number of jobs, n, and the objective function they consider are given in Table 2.

Looking at the results recorded in Table 2, it is clear that HHS$_{CA}$ is superior to HHS. However, for a solid conclusion, Friedman's non-parametric statistical test, [33], to compare the results of HHS vs HHS$_{CA}$ has been performed on each set of instances. The hypothesis tested was that the effect on the quality of solutions due to HHS and HHS$_{CA}$ is the same. If this is the case, then there is no difference.

Table 2: Mean f_i results

method	n	$f_1 \times 10^{-3}$	$f_2 \times 10^{-5}$	$f_3 \times 10^{-5}$	$f_4 \times 10^{-5}$	$f_5 \times 10^{-2}$	$f_6 \times 10^{-5}$	$f_7 \times 10^{-3}$	f_8
HHS	20	52.24	17.32	43.67	42.52	66.51	31.66	37.48	920
HHS_{CA}		52.20	17.22	42.34	41.41	64.98	31.02	37.35	860
HHS	40	50.70	16.93	34.52	31.71	51.78	31.11	30.01	946
HHS_{CA}		50.66	16.83	32.54	30.47	49.62	30.57	30.01	899
HHS	60	49.73	15.39	40.94	39.20	62.51	26.74	35.70	610
HHS_{CA}		49.48	14.93	38.04	36.80	59.47	25.90	35.24	529
HHS	80	45.69	14.33	32.07	29.07	51.18	26.22	27.64	545
HHS_{CA}		45.33	13.94	29.23	26.75	46.90	25.69	27.24	501

Table 3: p values of the Friedman's non parametric statistical test

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
$n = 20$	0.8348	0.0	0.0	0.0	0.0	0.0007	0.0025	0.0002
$n = 40$	0.8185	0.0	0.0	0.0	0.0	0.0001	0.8348	0.0013
$n = 60$	0.0186	0.0	0.0	0.0	0.0	0.0000	0.0011	0.0011
$n = 80$	0.0588	0.0	0.0	0.0	0.0	0.0004	0.2393	0.5316

If, on the other hand, there is no evidence to sustain such a conclusion, then the result favours HHS_{CA} given that its mean was consistently superior to that of HHS (as it can be appreciated from Table 2). The p values of every test are presented in Table 3. Whenever $p < 0.05$ the effects due to HHS and HHS_{CA} are different. Note that in most cases the hypothesis that both algorithms have similar effects is rejected in favour of HHS_{CA}. Since the only difference between HHS and HHS_{CA} is in the use or otherwise of the competence and affinity information, this is evidence that the inclusion of this information does have a positive effect on the solution quality observed in the solutions produced by HHS_{CA}.

4.2 A comparison of HHS_{CA} against established algorithms

We consider a number of solution approaches to HFS on real-world instances of the problem described in [15]. Beside HHS, we consider h_1GA which schedules stage 1 using GA and stages $2, ..., m$ using h_1; RKGA, the Random Keys Genetic Algorithm, [16], which uses a special representation of GA, Random Keys, to schedule stage 1 of the flowshop and h_1 to schedule the rest of the stages; GA_H, the Ruiz and Moroto's Genetic Algorithm, [17], and EDD-W, the Earliest Due Date and Weights (EDD-W) first, dispatching rule, [15], which is a simple dispatching

Table 4: Mean f_i results, $\frac{1}{10}\sum_{\psi=1}^{10} f_i(a(\psi))$

method	n	$f_1 \times 10^{-3}$	$f_2 \times 10^{-4}$	$f_3 \times 10^{-2}$	$f_4 \times 10^{-3}$	f_5	f_6	$f_7 \times 10^{-4}$	$f_8 \times 10^{-3}$
HHS	30	23.62	26.22	82.77	101.08	3.40	12.80	46.27	101.08
HHS_{CA}		23.28	25.72	82.27	69.77	3.00	7.60	44.92	69.48
h_1GA		23.18	25.75	82.83	70.20	3.00	7.40	45.02	70.20
RKGA		23.32	26.18	84.48	73.90	3.80	8.60	45.79	73.90
GA_H		23.43	26.58	83.27	77.18	0.00	8.20	45.08	77.18
EDD-W		31.20	36.60	149.30	138.54	16.40	24.40	58.46	138.54
HHS	60	44.50	80.20	193.87	407.08	7.20	33.00	107.50	407.08
HHS_{CA}		41.84	78.17	164.42	231.16	6.80	16.50	103.48	234.59
h_1GA		42.27	78.17	171.34	229.60	6.80	16.10	103.81	229.60
RKGA		43.34	81.31	189.99	285.64	8.60	23.30	108.91	285.64
GA_H		43.05	84.08	183.62	301.35	2.70	21.20	101.90	301.35
EDD-W		53.53	139.40	268.08	1095.78	58.90	117.20	219.53	1095.78
HHS	90	59.28	158.93	254.01	1012.89	16.90	59.10	205.06	1012.89
HHS_{CA}		55.51	151.13	212.43	441.62	10.50	25.70	170.68	437.55
h_1GA		56.46	151.29	225.56	449.66	10.40	26.50	171.44	449.66
RKGA		57.45	160.62	247.92	588.06	15.20	37.20	182.48	588.06
GA_H		56.82	167.67	238.02	647.22	4.60	36.50	160.85	647.22
EDD-W		66.28	285.41	300.43	2503.65	90.20	180.70	421.45	2503.65
HHS	120	80.86	283.39	344.06	1522.10	26.40	75.00	317.97	1522.10
HHS_{CA}		77.46	262.26	280.27	530.13	13.40	26.70	242.20	547.70
h_1GA		78.45	263.31	307.68	551.07	13.50	27.40	244.88	551.07
RKGA		79.39	284.98	339.62	788.96	23.80	49.90	275.91	788.96
GA_H		78.78	298.24	322.32	880.72	6.70	44.80	214.92	880.72
EDD-W		88.56	513.81	374.75	3771.84	123.20	228.50	724.44	3771.84

$f_i(a(\psi))$: f_i value of the solution provided by a to problem instance ψ; $a \in \{HHS, \ldots, \text{EDD-W}\}$

rule used in the practical situation which provided the test problems. The rule says "whenever a machine is free, the released job with the shortest due date is assigned to it. Ties are broken by preferring more important jobs, i.e. jobs with the highest weights."

Four types of problems with 30, 60, 90 and 120 jobs (10 of each) were solved. Details on the way the problems have been generated from the real data can be found in [15].

The results recorded in Table 4 show a clear superiority of HHS_{CA} over the rest of the algorithms on most problems. GA_H seems to do a bit better on f_5 and f_7, otherwise RKGA, GA_H as well as HHS and EDD-W, the dispatching rule used in practice by the company that supplied the data, are outperformed everywhere by HHS_{CA}. h_1GA also performed well in the few cases of f_1 and f_6 for $n = 30$ and f_4 and f_6 for $n = 60$. Again, it is otherwise surpassed by HHS_{CA} everywhere else too. So, there is no doubt that HHS_{CA} is superior to all algorithms considered. The superiority is so clear that there is no real need for any statistical analysis.

5 Conclusion and further work

The paper considered the case of enhancing hyper-heuristics that use GA as the coordinating meta-heuristic by incorporating in their solutions, which are sequences of low level heuristics, information related to these low level heuristics. This information concerns

- how good individual low level heuristics are at solving a given problem (competence);
- how good they are at working together to solve the problem (affinity).

Models of the competence and affinity are devised and then shown how they can be implemented to improve the quality of solutions to the problem. Algorithms to that effect have been given.

A large number of randomly generated test problems of the notoriously hard HFS problem have been solved with both the pure HHS, i.e. the HHS that does not include the competence and affinity information and the HHS_{CA} which includes the information. Moreover, a set of real-world instances has also been solved with HHS, HHS_{CA} as well as a host of other well established algorithms such as RKGA, GA_H, h_1GA and EDD-W. Results in terms of the quality of the solution obtained have been recorded. These show beyond any doubt that, indeed competence and affinity information improves the quality of the solutions of the instances considered. Since real world problems often have characteristics that randomly generated ones do not, the rather impressive performance of HHS_{CA} on them compared to the rest of the algorithms considered, establishes beyond doubt that the idea of including competence and affinity information when low level heuristics are used is a potent one.

Note that the approach is reminiscent of the so called memetic and meta-Lamarckian algorithms, [34, 35]. As such, the idea of competence and affinity may be extended to full blown algorithms working under a meta-heuristic or a coordinator algorithm, which is not necessarily a search algorithm in itself, within a cooperative/competitive framework, [36, 37].

References

[1] Peter Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. K. Burke and W. Erben, editors, *Proceedings of the Third International Conference on the Practice and Theory of Automated Timetabling III, Lecture Notes in Computer Science, Vol. 2079*, pages 176–190. Springer-Verlag, 2000.

[2] Eric Soubeiga. *Development and Application of Hyperheuristics to Personnel Scheduling*. PhD thesis, School of Computer Science and Information Technology, The University of Nottingham, 2003.

[3] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyperheursitics: A survey of the state of the art. Technical report, School of Computer Science and Information Technology, University of Nottingham, UK, 2010.

[4] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcani, and J. R. Woodward. Exploring hyper-heursitic methodologies with genetic programming. In C. Mumford and L. Jain, editors, *Computational Intelligence: Collaboration, Fusion and Emergence*. Springer, 2009.

[5] E. Burke and E. Soubeiga. Scheduling nurses using a tabu-search hyperheuristic. In *Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2003)*, 2003.

[6] Peter Cowling, Graham Kendall, and Limin Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of Congress on Evolutionary Computation (CEC2002)*, pages 1185–1190. IEEE, 2002.

[7] Edmund Burke, Graham Kendall, Dario Landa Silva, Ross O'Brien, and Eric Soubeiga. An ant algorithm hyperheuristic for the project presentation scheduling problem. In *Proceedings of the Congress on Evolutionary Computation (CEC 2005)*, pages 2263–2270. IEEE press, 2005.

[8] Masri Ayob and Graham Kendall. A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In *Proceedings of the International Conference on Intelligent Technologies, InTech'03*, pages 132–141, 2003.

[9] S. Petrovic and R. Qu. Case-based reasoning as a heuristic selector in a hyper-heuristic for course timetabling problems. In *Proceedings of the 6th International Conference on Knowledge-Based Intelligent Information Engineering Systems and Applied Technologies (KES'02)*, pages 336–340, 2002.

[10] E. K. Burke, B. L. McCarthy, S Petrovic, and R. Qu. Knowledge discovery in a hyper-heuristic for course timetabling using case-based reasoning. In *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), Lecture Notes in Computer Science, Vol. 2740*, pages 276–286. Springer-Verlag, 2002.

[11] Sanja Petrovic, Carole Fayad, and Dobrila Petrovic. Job shop scheduling with lot-sizing and batching in an uncertain real-world environment. In Graham Kendall, L. Lei, and Michale Pinedo, editors, *Proceedings of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2005)*, pages 363–379, New York, USA, 2005.

[12] Emma Hart and Peter Ross. A heuristic combination method for solving job-shop scheduling problems. In *Lecture Notes in Computer Sciences (1498)*, pages 845–854. Springer-Verlag, 1998.

[13] Hsiao-Lan Fang, Peter Ross, and Dave Corne. A promising hybrid GA/Heuristic approach for open-shop scheduling prblems. In A. Cohn, editor, *11th European Conference on Artificial Intelligence (ECAI 94)*, pages 590–594. John Wiley & Sons, Ltd., 1994.

[14] Edmund Burke, Moshe Dror, Sanja Petrovic, and Rong Qu. Hybrid graph heuristic within a hyper-heuristic approach to exam timetabling problems. In B. L. Golden, S. Raghavan, and E. A. Wasil, editors, *Proceedings of the 9th Informs Computing Society Conference*, pages 79–91. Springer, 2005.

[15] J. A. Vázquez Rodríguez. *Meta-Hyper-Heuristics for Hybrid Flow Shops*. PhD thesis, Department of Mathematical Sciences, University of Essex, Colchester CO4 3SQ, UK, 2007.

[16] Fouad Riane, Abdelhakim Artiba, and Salah E. Elmaghraby. Sequencing a hybrid two-stage flowshop with dedicated machines. *International Journal of Production Research*, 40:4353–4380, 2002.

[17] Rubén Ruíz García and Concepción Maroto. A genetic algorithm for hybrid flow shops with sequence dependent setup times and machine elegibility. *European Journal of Operational Research*, 169:781–800, 2006.

[18] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1992.

[19] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Massachussetts, 1989.

[20] J. N. D. Gupta. Two-stage hybrid flow shop scheduling problem. *Operational Research Society*, 39:359–364, 1988.

[21] J. A. Hoogeveen, J. K. Lenstra, and B. Veltman. Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. *European Journal of Operational Research*, 89:172–175, 1996.

[22] Z. H. Jin, K. Ohno, T. Ito, and S. E. Elmaghraby. Scheduling hybrid flowshops in printed circuit board assembly lines. *Production and Operations Management*, 11:216–230, 2002.

[23] H. D. Sherali, S. C. Sarin, and M. S. Kodialam. Models and algorithms for a two-stage production process. *Production Planning & Control*, 1:27–39, 1990.

[24] A. G. Guinet. Textile production systems: A succession of non-identical parallel processor shops. *Journal of the Operational Research Society*, 42:655–671, 1991.

[25] J. Grabowski and J. Pempera. Sequencing of jobs in some production system. *European Journal of Operational Research*, 125:535–550, 2000.

[26] E. H. Aghezzaf and H. Van Landeghem. An integrated model for inventory and production planning in a two-stage hybrid production system. *Internation Journal of Production Research*, 40:4323–4339, 2002.

[27] A. Allahverdi and F. S. Al-Anzi. Scheduling multi-stage parallel-processor services to minimize average response time. *Journal of the Operational Research Society*, 57:101–110, 2006.

[28] Chen Lu, Xi Li-Feng, Ca I Jian-Guo, Bostel Nathalie, and Dejax Pierre. An integrated approach for modeling and solving the scheduling problem of container handling systems. *Journal of Zhejiang University SCIENCE A*, 7:234–239, 2006.

[29] H. T. Lin and C. J. Liao. A case study in a two-stage hybrid flow shop with setup time and dedicated machines. *International Journal of Production Economics*, 86:133–143, 2003.

[30] F. Choong, S. Phin-Amnuaisuk, and M. Y. Alias. Metaheuristic methods in hybrid flow shop scheduling problem. *Expert Systems with Applications*, 38(9):10787–10793, 2011.

[31] Michael Pinedo. *Scheduling Theory, Algorithms and Systems*. Prentice Hall, 2002.

[32] José Antonio Vázquez Rodríguez and Abdellah Salhi. A robust meta-hyperheuristic approach to hybrid flow shop scheduling. In K. Dahal and P. Cowling, editors, *Evolutionary Scheduling*, pages 125–142. Springer, 2007.

[33] M. Hollander and D. A. Wolfe. *Nonparametric Statistical Methods*. Wiley, 1973.

[34] P. Moscato and C. Cotta. *A Gentle Introduction to Memetic Algorithms*, pages 105–144. Kluwer Academic Publishers, 2003. In: Handbook of Metaheuristics.

[35] Y. S. Ong and A. J. Kean. Meta-Lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):99–110, 2004.

[36] A. Salhi and Ö. Töreyen. *A Game Theory-based Multi-Agent System for Expensive Optimisation*, pages 212–232. Springer-Verlag Berlin Heidelberg, 2010. Chapter 9 of Computational Intelligence in Optimization-Applications and Implementations, Y. Tenne and C.-K Goh editors.

[37] Ö. Töreyen. A game theory-based multi-agent system for solving complex optimisation problems. Master's thesis, Department of Mathematical Sciences, University of Essex, Colchester CO4 3SQ, UK, 2008.