An Ant Algorithm Hyperheuristic for the Project Presentation Scheduling Problem

Edmund Burke, Graham Kendall, Dario Landa Silva and Ross O'Brien
Automated Scheduling, optimisAtion and Planning (ASAP) Research Group
School of Computer Science and Information Technology,
University of Nottingham, Jubilee Campus, Wollaton Road
Nottingham, NG8 1BB
ekb, gxk, jds, rob @ cs.nott.ac.uk

Eric Soubeiga
Business Modelling Group
KPMG LLP
1-2 Dorset Rise
London, EC4Y 8EN
eric.soubeiga @ kpmg.co.uk

Abstract- Ant algorithms have generated significant research interest within the search/optimisation community in recent years. Hyperheuristic research is concerned with the development of *"heuristics to choose heuristics"* in an attempt to raise the level of generality at which optimisation systems can operate. In this paper the two are brought together. An investigation of the ant algorithm as a hyperheuristic is presented and discussed. The results are evaluated against other hyperheuristic methods, when applied to a real world scheduling problem.

1 Introduction

Hyperheuristic research has emerged in recent years as an approach to increase "the level of generality at which optimisation systems can operate" [1]. This increase in the level of generality could underpin a new generation of decision support systems which are applicable across a range of problems rather than the current state of the art which tends to focus on bespoke tailor-made systems.

We can achieve this by using a framework with a built-in level of abstraction. A set of "low-level" heuristics are developed which interact with a given solution to a problem, and explore its neighbourhood in the solution space. A higher level search method, termed a hyperheuristic, manages "the choice of which lower-level heuristic method should be applied at any given time, depending on the characteristics of the region of the solution space currently under exploration" [11].

Recent hyperheuristic research has focused on exploring the use of existing techniques and metaheuristics as hyperheuristics. Terashima-Marin et al. developed a constructive hyperheuristic based on the genetic algorithm technique to find indirect ways to solve exam timetabling problems [26]. Ross et al. developed constructive hyperheuristics using learning classifier systems and genetic algorithms respectively in order to learn, for a given stage of bin-packing problems, which heuristics were more useful than others [22, 23]. Genetic algorithm approaches have also been explored by Cowling et al. [10] and Han and Kendall [18, 19]. Other approaches have included tabu search (Burke et al. [3]), and simulated annealing (Dowsland et al. [16]). A graph-based hyperheuristic (Burke et al. [6]) and a custom choice function hyperheuristic (Cowling et al. [11, 12], Gaw et al. [17] and Soubeiga [25]) have been developed for a variety of optimisation problems. Continued work has developed the choice function approach for use with multi-objective problems [5].

Burke et al. [4, 7] and Petrovic and Qu [20] developed a case-based hyperheuristic approach to timetabling problems which suggested heuristics to use based on their performance in previous similar situations which were stored in a case base.

A more detailed overview of hyperheuristic methods can be seen in [1].

As far as the authors are aware, Ant Colony Optimisation has not been explored elsewhere within the context of hyperheuristics. The approach was introduced by Dorigo [13] and was inspired by the behaviour of real ants in nature.

Ant colony techniques have been successfully applied to many problems including the Travelling Salesman Problem [14], Quadratic Assignment Problem [14], Job-Shop Scheduling Problems [15], generic constraint satisfaction problems [21], University Course Timetabling Problems [24], cutting and packing [2] and graph colouring problems [9]. The standard technique is a constructive one: A colony of ants begins with no solutions. Each ant constructs a solution by making decisions stochastically, using existing problem constraints and heuristics combined with experience (which is analogous to a substance called pheromone). The colony then reinforces decisions in the construction process according to their successes by adding pheromone, which also decays to mitigate against poorer decisions.

Our purpose in this paper is to investigate the ant algorithm technique as a means of constructing effective sequences of heuristic moves.

Some parallels can perhaps be drawn with the genetic algorithm hyperheuristic technique [10, 18, 19, 23, 26]. This constructs a population of chromosomes in which genes represent heuristics, and therefore chromosomes represent sequences of heuristics. Our ant algorithm hyperheuristic involves a population of ants each of which produces a sequence of heuristics stochastically.

We may also draw parallels from an unusual application of ants to the graph colouring problem [8, 9]. In [9] a standard ant colony constructs complete solutions and learns from its group efforts as to how to create better solutions. However, in [8] a colony of ants is released upon an existing (and not necessary feasible) solution and traverses the graph, making local-search repair efforts to improve it. In the same way, we equip a colony of ants with pre-constructed solutions and let them collectively learn about

0-7803-9363-5/05/$20.00 ©2005 IEEE.

the heuristic space, using this knowledge to guide their selection of appropriate low level heuristics to improve their given solutions.

We test our ant algorithm hyperheuristic on the Project Presentation Scheduling Problem which is described in section 2. The algorithm itself is described in section 3. Section 4 presents our results and analysis of our experiments, with comparisons to other hyperheuristics, and conclusions are presented in section 5.

2 The Project Presentation Scheduling Problem

The Project Presentation Scheduling Problem is introduced in [12]. Students on the Computer Science course at the University of Nottingham are required to undertake a supervised year-long project as part of their undergraduate final year. The problem is concerned with the scheduling of an individual ten-minute presentation (with five minutes for questions) which is given before at least three members of academic staff (known as the 1st Marker, the 2nd Marker and the Observer), preferably one of whom is the supervisor of the project. Presentations are grouped into hour-long sessions. The problem was formulated and hyperheuristics applied to the task of assigning presentations to sessions in available rooms with the three members of academic staff with the (soft constraint) aims of distributing an equal number of presentations, sessions and *bad* sessions (i.e. the 9-10am and 5-6pm sessions) to each academic staff member, maximising the number of students assessed by their own supervisor and also maximising the amount of interest the trio of staff members would have in the presentations they have to mark in a particular session.

Three real-life instances of this problem are available at http://www.cs.nott.ac.uk/~rob/ppsproblem.html, which increase in difficulty, labelled csit0, csit1 and csit2 [12, 25]. In csit0 there are 151 students, 26 staff members, 80 available sessions and 2 rooms; in csit1 there are 240 students, 24 staff members, 36 sessions and 2 rooms, and csit2 is the same as csit1 but with two staff members declared absent during the presentation timetable. In the latter problems (csit1 and csit2) the constraints are tighter, with more presentations to fit into less rooms and less staff members being available.

A feasible solution is created, and the hyperheuristic makes use of eight iterative heuristics to explore the solution space thereafter:

- h_1 Replace one random lecturer j_1 in a random session in which he/she is scheduled for presentations with a second random lecturer j_2.
- h_2 The same as h_1 but j_1 has the largest number of scheduled sessions.
- h_3 The same as h_2 but the session chosen is the one where j_1 has the smallest number of presentations.
- h_4 Move a random presentation i from its current room-session assignment to another.
- h_5 The same as h_4 but presentation i is that for which the sum of presentations involving all three involved lecturers is smallest of all sessions.
- h_6 The same as h_5 but the new session is one in which at least one of the involved lecturers is already scheduled to mark presentations.
- h_7 Swap the 2nd marker of one presentation with the observer of another (a supervisor may not be removed).
- h_8 Swap the 1st marker of one presentation with the 2nd marker of another (a supervisor may not be removed).

The problem is formulated as a minimisation problem. This formulation, the constructive heuristic used to create the initial feasible solution and the eight iterative heuristics presented here are all the same as those used in [12, 25].

3 Methodology

We create a network in which vertices represent heuristics and directional transitional arcs exist between heuristics if it is possible to apply one immediately after the other. We then create a number of ants, each of which represents a *hyperheuristic agent* supplied with an initial solution in the solution space and access to the heuristics and evaluation functions. The ants are scattered uniformly among the vertices of the network.

The ants then construct a sequence of heuristics (a *path*) by traversing the network. At each decision point, each ant selects the next vertex it will visit, traverses the arc to that vertex, and applies the heuristic represented by that vertex to its current solution. Vertices and arcs may freely recur within the path.

After each ant has visited a certain number of heuristics, the ant pauses to analyse the path it has just traversed and to lay an amount of pheromone on each edge in that path according to the improvement in the quality of the solution during the entire path. Each ant proceeds to generate its next path. We term the time taken between all ants beginning their paths and all ants completing their paths as a *cycle*, and the algorithm continues for as many cycles as is required.

In the standard ant algorithm each successive vertex j is selected from an ant on vertex i using a probability calculated by employing the pheromone value on the arc from vertex i to vertex j, which we will write as τ_{ij}, and some form of heuristic information known in advance (in the Travelling Salesman Problem this is called *visibility*) about vertex j, which we will write η_j (illustrated in Figure 1). Since the hyperheuristic method has no knowledge of each low-level heuristic's potential in advance, and since this potential will vary as the colony traverses the solution space, the visibility function must be initially impartial and continually adaptive.

At the end of each cycle all of the ants relocate in the solution space to the best solution found during that cycle. This step does not involve updating visibility or pheromone information but is instead specifically intended to restrict exploration and keep the ants in roughly the same area of

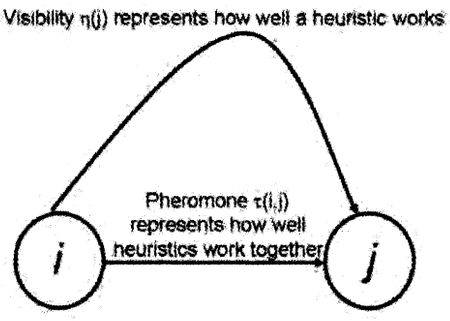

Figure 1: The information available at each decision point.

the solution space during a cycle in order to keep the collective visibility and pheromone information as relevant as possible.

For the same reason, the system starts again at the vertex in the network whose heuristic discovered that best solution, i.e. if heuristic h_x causes an ant to find the best solution of the cycle, all ants will begin the next cycle at vertex x, in order that if the first heuristic h_y of an ant's new tour produces an improvement, pheromone will be laid on the arc (x, y).

At the decision level an ant may choose to reject a new solution it discovers if it is poorer than the ant's current solution. If a solution is rejected, we punish the visibility of the heuristic which caused the detriment of solution quality but for the purposes of laying pheromone we ignore the arc. That is, if an ant performs heuristics h_x and h_y, and h_y leads to a worse solution, the solution is rejected and the visibility of heuristic h_y is punished. If a third heuristic h_z is chosen and leads to a better solution, pheromone will be laid on edge (x, z) and not (x, y) or (y, z).

Without this criterion the ant algorithm hyperheuristic is effectively an Any Moves (AM) Hyperheuristic, i.e. the hyperheuristic accepts any heuristic move, regardless of any improvement, at the decision level. With this criterion the ant algorithm hyperheuristic is an Only Improving (OI) Hyperheuristic, i.e. at the decision level the hyperheuristic accepts only new solutions which are better than the current solution. Previous research [25] indicates that OI hyperheuristics may be more restricted and more likely to be trapped within local optima, while at the decision level there is nothing to prevent AM hyperheuristics from exploring areas of the solution space of progressively lower quality.

The ant algorithm hyperheuristic is outlined in the following pseudo-code:

1. **Initialise**

   Set $t := 0$. $\{t$ is the heuristic calls counter$\}$.

   Set $LP := 0$. $\{LP$ is the length of a path$\}$.

   For every vertex j set an initial value $\eta_j = 0$.

   Scatter the m ants uniformly on the n vertices.

   Initialise a solution S, and best solution, S_b.

   For $k := 1$ to m do

   - Provide a copy of solution S, S_k, to each ant.
   - Apply heuristic j, where vertex j is ant k's current location, to S_k.
   - Update the kth ant's location in the solution space to the resulting S'_k, and update η_j according to equation 1.

   Set $t := t + m$.

   For every edge (i, j) set an initial value $\tau_{ij}(t) = 0$.

2. **Heuristic Exploration**

   For $k := 1$ to m do

   - Choose the heuristic j for ant k to move to, with probability $probability_{ijk}(t)$ given by equation 7 $\{$at decision point t the kth ant is on vertex $i\}$.
   - Apply heuristic j to solution S_k to produce S'_k.
   - If (Only Improvement & S'_k is worse than S_k) then Reject S'_k.
     else Accept S'_k.
   - If (S'_k is accepted)
     - Move ant k to vertex j.
     - Set $S_k := S'_k$.
   - If (S'_k is better than S_b)
     - Set $S_b := S'_k$.

3. **Visibility Update**

   - For $k := 1$ to m do
     - Update η_j according to equation 1.
   - Set $LP := LP + 1$.
   - Set $t := t + m$.

4. **Pheromone Update**

   - If ($LP = n$)
     - Set $LP := 0$.
     - Update $\tau_{ij}(t)$ for all accepted arcs (i, j) according to equation 2.
     - Set $S := S_b$ $\{S_b$ is the best solution found during this cycle).

5. **Stopping Condition**

   - If ($t < t_{max}$)
     - Goto step 2.

   else
     - Output best solution S_b.
     - Stop.

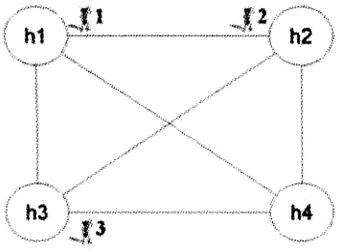

Figure 2: Three ants are distributed among four vertices, each with a copy of an initial solution S. To provide an initial visibility value, each ant applies the heuristic corresponding to their location to their copy of S.

The overall approach is illustrated in Figures 2, 3, 4 & 5.

We use a visibility function inspired by the choice function hyperheuristic [11, 12, 25], which uses information based on solo and sequential performance of the different heuristics. As pheromone corresponds to the sequential performance, we use a visibility function η_j corresponding to heuristic j's individual performance, and update this value after all ants have completed their moves:

$$\eta_j(t) = \gamma\eta_j(t-m) + \sum_{k}^{m} \frac{I_{kj}(t)}{T_{kj}(t)} \qquad (1)$$

where m is the number of ants in the colony (i.e. the number of heuristic calls made since the last update)$I_{kj}(t)$ is the improvement produced by heuristic j on ant k's current solution at decision point t (which could be negative), $T_{kj}(t)$ is the number of CPU seconds heuristic j took to run on ant k's current solution at decision point t and γ is a constant weight valued between 0 and 1 which emphasises recent performance as a weight emphasising recent performance. (The notation α is sometimes used (e.g. [25]) within the solo factor of the choice function. However, since both the choice function hyperheuristic and ant algorithm technique make use of the symbols α and β to weigh contributing aspects, we use γ for the aspect of the choice function hyperheuristic given here and use α and β for aspects of the ant algorithm technique given below.)

Our visibility function draws upon the fact that all ants moving to a specific vertex or traversing a specific arc add their visibility contributions together and with equal weight.

The choice function hyperheuristic [25] also usually includes a diversification contributor which encourages the use of heuristics which have not been recently used. Since we use several ants, a probabilistic selection procedure and not many heuristics (in this case, eight) we have anticipated that there will already be sufficient diversity.

The ants share their confidence in the sequences of heuristics using pheromone, which also decays to clear away older preferences and emphasise recent performance of low-level heuristics. Once all ants have completed their paths (i.e. when the cycle is completed; there are n heuristics and therefore n heuristic calls in a path for each ant, so this occurs every $m \cdot n$ heuristic calls), the amount of

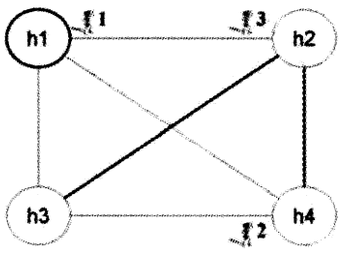

Figure 3: Ant 1 chooses to remain at vertex/heuristic h_1, Ants 2 and 3 respectively choose to move to vertices/heuristics h_4 and h_2. The ants apply these heuristics to their respective current solutions and update the visibility values of heuristics h_1, h_2 and h_4.

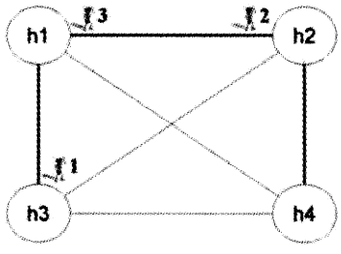

Figure 4: Ants 1, 2 and 3 respectively choose to move to vertex/heuristics h_3, h_2 and h_1. As in Figure 2, visibility values for these three heuristics are updated.

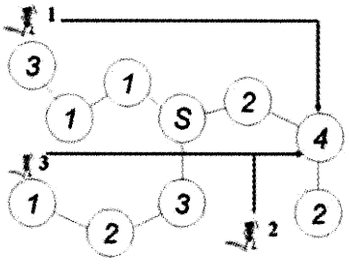

Figure 5: A representation of the solution space, with S representing the ants' solution at the beginning of the cycle, and each ant having traversed their own path. Pheromone will now be laid on the arcs $(1,1)$, $(1,3)$, $(2,4)$, $(4,2)$, $(3,2)$ and $(2,1)$ in proportion to the improvement between the final solutions and S. Assuming the best solution of the cycle was discovered as Ant 2 applied heuristic h_4, the ants will adopt this best solution, which will be the new S in the next cycle.

pheromone on each arc (denoted by $\tau_{ij}(t)$ for the arc between heuristic i and heuristic j at decision point t) is adjusted as follows:

$$\tau_{ij}(t) = (1-\rho)\tau_{ij}(t - m \cdot n) + \sum_{k}^{m} \frac{\#_{ij}(P_k(t)) \cdot I(P_k(t))}{T(P_k(t))} \quad (2)$$

where ρ is the pheromone evaporation coefficient, $P_k(t)$ is the path ant k traversed during the cycle ending at decision point t, $\#_{ij}(P_k(t))$ is the number of times the arc (i, j) occurs during path $P_k(t)$, $I(P_k(t))$ is the improvement produced by the heuristics ant k used during its last path (i.e. the difference between the best solution quality found during this path and the best solution quality found at the end of the previous cycle), and $T(P_k(t))$ is the duration of that path in CPU seconds. Thus, for a given ant's path, an arc traversed twice in that path receives twice the amount of pheromone as an arc traversed once in the same path.

The ant's actual decision-making process requires the algorithm to combine the visibility and pheromone values for each of the arcs the ant could potentially traverse into a single positive value, in order to be properly used in a roulette probability system. This becomes an issue when it is possible for one or more heuristics to find a solution of poorer quality to the current solution, whether to escape local optima or otherwise encourage a diverse search of the solution space. The issue is concerned with negative values of η_j or τ_{ij}. Our conversion process is borrowed from the choice function variant "RouletteFunction" [11]. We first calculate a value V for each heuristic j, from the previous heuristic i, using the formula:

$$V_{ij}(t) = \alpha\eta_j(t) + \beta\tau_{ij}(t) \quad (3)$$

From this we calculate a positive value (PV) using the formula

$$PV_{ij}(t) = max\{V_{ij}(t), Q\sigma^{V_{ij}(t)}\} \quad (4)$$

where

$$Q = \frac{\sum_h max\{0, V_{ih}(t) + \epsilon\}}{10 \cdot n} \quad (5)$$

(where $h \in H$, the set of low-level heuristics, and $n = |H|$), and ϵ and σ are constants included to ensure that poor-performing heuristics have a small non-zero probability of being selected, proportionally enhanced by Q if other heuristics are performing particularly well. We set ϵ to 0.001 to give a small boost to heuristics whose value V = 0, and σ to 1.001 to ensure that is a monotonic conversion of V values to positive V values: negative V values to the range 0 to 1, zero values to 1, and positive values to the range 1 to 8.

We include two further safeguards to promote the choosing of heuristics and to promote the exploration of arcs which may not yet have been selected:

If arc (i, j) has not yet been selected, i.e. $PV_{ij}(t) = 0$, we temporarily assign to $PV_{ij}(t)$ the value of $PV_{ih}(t)$, where $h \in H$ and (i, h) is the current highest-ranking arc in the set of arcs beginning at vertex i and having been previously selected.

$$PV_{ij}(t) = max\{max_{h \in H} PV_{ih}(t), 0\} \quad (6)$$

If none of the heuristics are performing well, i.e. $Q = 0$, we set all PV values to 1, such that all heuristics have an equal probability of selection. Finally, the probability of any arc (i, j) being selected is

$$probability_{ijk}(t) = \frac{PV_{ij}(t)}{\sum_{h \in H} PV_{ih}(t)} \quad (7)$$

4 Experiments and Results

All algorithms were coded in Microsoft Visual C++ version .NET 2003 and all experiments were run on a PC Pentium IV 1.8GHz with 256MB RAM running under Microsoft Windows 2000 version 5. The stopping condition used in [12, 25] was 600 seconds of CPU time on PC Pentium III 1.0GHz with 128MB RAM. This equated to approximately 1000 heuristic calls, which we use as our stopping condition; this equates to approximately 30-40 seconds using our current framework.

We use the same constructive heuristic as in [12, 25], which includes random elements. In order to keep our work consistent with the results in these papers, 50 solutions were created for each of the three problem instances and the 5 nearest to the initial solution in [12, 25] selected.

As we are using 8 low level heuristics in the Project Presentation Scheduling Problem, our ant algorithm hyperheuristics use a graph with 8 vertices. Our corresponding path length is 8 heuristic calls. We consider the number of ants to be a parameter, and we experiment with between 3 and 5 ants. This range is deliberately smaller than might typically be considered for an approach based on Ant Colony Optimisation. The necessity of this range is that we understand our exploration of the solution space to be unpredictably dynamic and non-deterministic. Our values are chosen to provide enough ants to demonstrate a collective behaviour but not so many that we have too few cycles with which to learn within the stopping condition of 1000 heuristic calls.

Every hyperheuristic was run 10 times, twice using each initial solution, on each of the three problem instances described in section 2. In all ant algorithm hyperheuristics, each ant begins with the same initial solution.

The hyperheuristics are the Choice Function Hyperheuristic (CF) and a Simple Random Hyperheuristic (SR) presented in [12, 25] for comparative purposes and our Ant Algorithm Hyperheuristic with 3-5 ants (AA3-AA5). The hyperheuristics were also distinguished by their solution acceptance criteria: either the hyperheuristic accepts Any Move (AM), or it accepts only improving moves (OI).

We assign to α, β, γ respectively the values 0.7, 0.7, 0.7, since early experiments did not show any discernable patterns but 0.7 appeared to be generally good.

The results presented in Table 1 show the worst, mean average and best results from each algorithm applied to the three problem instances. The mean initial solution quality is supplied at the top of the respective problem instance's column. The best result from each column is highlighted in bold.

It is immediately evident that in the Any Moves (AM)

Table 1: Results of hyperheuristic experiments (best results for each column are in bold).

		Instance: *csit0* Initial solution quality: -895.5			Instance: *csit1* Initial solution quality: -2428.4			Instance: *csit2* Initial solution quality: -940.39		
		Worst	Average	Best	Worst	Average	Best	Worst	Average	Best
CF	(AM)	-1421.1	-1515.17	-1620.48	-2524.6	-2715.79	-2894.8	-1179.6	-1403.5	-1529.25
CF	(OI)	-1074.37	-1555.46	-1669.64	-2854.6	-2959.85	-3058.7	-1650.6	-1712.19	-1794
SR	(AM)	-1105.5	-1346.91	-1503.18	-2372.2	-2542.84	-2712.6	-1039.4	-1194.35	-1351.2
SR	(OI)	-1598.08	**-1672.44**	**-1757.39**	-2861.3	-2999.02	**-3062.3**	-1661.4	-1733.76	-1783.15
AA3	(AM)	-1547.28	-1628.31	-1669.39	-2816.15	-2903.66	-3001.8	-1566.85	-1655.39	-1721.95
AA4	(AM)	-1541.6	-1589.54	-1657.05	-2827	-2912.9	-3010.6	-1560.15	-1642.98	-1725.6
AA5	(AM)	-1534.53	-1594.32	-1679.81	-2838.1	-2913.53	-2998.6	-1634.6	-1669.93	-1698.8
AA3	(OI)	**-1639.2**	-1665.02	-1707.29	**-2943.75**	**-3004.55**	-3051.5	**-1691.3**	**-1758.1**	-1809.4
AA4	(OI)	-1577.75	-1645.53	-1705.16	-2894.2	-2993.17	-3049.8	-1676.8	-1754.37	-1788.8
AA5	(OI)	-1579.83	-1642.03	-1681.7	-2913.35	-2995.07	-3059.8	-1665.2	-1751.38	**-1835.45**

case, our Ant Algorithm Hyperheuristics outperform both the Simple Random and the Choice Function Hyperheuristics. In the Only Improvement (OI) case the Ant Algorithm Hyperheuristic is merely competitive, though it does perform better on average in two of the three problem instances. Specifically, in two of the three problem instances (csit1 and csit2) the Ant Algorithm Hyperheuristic with 3 ants performs better than the Ant Algorithm Hyperheuristics with 4 or 5 ants in the average case. However, in both problem instances the Ant Algorithm Hyperheuristic with 5 ants performs better than the Ant Algorithm Hyperheuristics with 3 or 4 ants in the best case, and in instance csit2 produces the best solution of all experiments.

We note that on average the Ant Algorithm Hyperheuristic results worsen as the number of ants increases. After each cycle the colony adopts the best solution found by the colony during that cycle. Each cycle is thus an exploration of the local area around the previous best solution. The more ants that are used, the broader and more intense that exploration becomes, and also the greater the number of solutions to select from. Larger colonies effectively consider the local problem for longer, and may make better decisions per cycle, but at the cost of making slower progress through the solution space. We speculate that in more difficult problem instances larger colonies may be preferable, but that in the problem instances explored here 3 ants are sufficient.

It was difficult to establish any other particular trends. During our experimentation we considered a number of visibility functions and ways of updating the pheromone levels, but no significant differences in results were reached. It is possible that this could be attributed to our stopping condition of 1000 heuristic calls. With 8 low-level heuristics there are 64 arcs upon which the ants can lay pheromone, and some heuristic calls will initially be used providing initial visibility information, so on average each arc will only be used 15.625 times per run, which may be too few to adequately learn the dynamics of the search space. Equally since each ant explores 8 arcs per cycle, there will only be $\frac{1000-8}{8 \cdot m}$ = 24.8-41.3 cycles, and so perhaps pheromone information is updated too infrequently to make a difference.

If this is the case then the colony may still be *inexperienced* by the end of the run, and the selection of each new low-level heuristic is still more significantly decided by random probabilities than by the functions which decide those probabilities. This would explain why the Simple Random Hyperheuristic (Only Improving) produced a solution better than that of the Ant Algorithm Hyperheuristics in instance csit0.

It can be noted, however, that despite this *inexperience*, the results of our experiments are superior to those published in [12, 18, 25].

5 Conclusions

We have introduced a new hyperheuristic technique which has been shown to be competitive or better compared against other hyperheuristics in the context of an established problem from the literature. We have also provided a basis for exploring the idea of using a population of "ant" hyperheuristic agents which learn from each other to improve their explorations of the solution and heuristic spaces.

The Ant Algorithm Hyperheuristic shows promise as a technique, and the work described in this paper also generates a number of future research directions. For example, a more effective visibility function might be explored. At present the method includes manually set parameters but it would be possible to explore self-adaptive parameters. Since the number of ants used by the hyperheuristic was shown to have an effect on the final solution quality, an investigation on the scalability of our approach, that is, to find the relation between the number of active hyperheuristic agents (ants) and the problem size would also appear to be a useful research direction.

Ongoing research will investigate each of these ideas in the contexts of a wider range of real-world problems.

Acknowledgements

We would like to thank David Redrup, who originally helped develop the Ant Algorithm Hyperheuristic concept; Julie Greensmith, for her support; Dr. Matthew Hardy and Dr. Steven Bagley, for assistance in programming, and

the EPSRC for sponsorship of this research (grant number GR/N36387/01).

Bibliography

[1] E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross and S. Schulenburg. Handbook of Metaheuristics, chapter 16, Hyperheuristics: An Emerging Direction in Modern Search Technology, pp. 457-474. Kluwer Academic Publishers, 2003.

[2] E. Burke and G. Kendall, Applying Ant Algorithms and the No Fit Polygon to the Nesting Problem, Proceedings of 12th Australian Joint Conference on Artificial Intelligence, Sydney, Australia, December 6-10, 1999, Lecture Notes in Artificial Intelligence vol. 1747, Foo, N. (Ed), pp. 453-464, 1999.

[3] E. Burke, G. Kendall and E. Soubeiga, A Tabu-Search Hyperheuristic for Timetabling and Rostering, Journal of Heuristics volume 9 issue 6, pp. 451-470, 2003.

[4] E. Burke, M. Dror, S. Petrovic and R. Qu, Hybrid Graph Heuristics in a Hyper-Heuristic Approach to Exam Timetabling, in The Next Wave in Computing, Optimization and Decision Technologies (eds. B.L. Golden, S. Raghavan & E.A. Wasil), pp. 79-92, Springer 2005.

[5] E. Burke, J.D. Landa Silva and E. Soubeiga, Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling, to appear in: Ibaraki T., Nonobe K., Yagiura M. (eds.), Meta-heuristics: Progress as Real Problem Solvers, Springer, 2005.

[6] E. Burke, A. Meisels, S. Petrovic and R. Qu, A Graph-Based Hyper Heuristic for Timetabling Problems. Accepted for publication in the European Journal of Operational Research, 2005.

[7] E. Burke, S. Petrovic and R. Qu, Case Based Heuristic Selection for Timetabling Problems. Accepted for publication in the Journal of Scheduling, 2005.

[8] F. Comellas and J. Ozn, An Ant Algorithm for the Graph Colouring Problem, ANTS'98 From Ant Colonies to Artificial Ants; in proceedings of the First International Workshop on Ant Colony Optimization, Brussels, 1998.

[9] D. Costa and A. Hertz, Ants can colour graphs, Journal of the Operations Research Society 48, pp. 295-305, 1997.

[10] P. Cowling, G. Kendall and L. Han, An investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. Proceedings of the Congress on Evolutionary Computation 2002, CEC 2002, pp. 1185-1190, 2002.

[11] P. Cowling, G. Kendall and E. Soubeiga, A Hyperheuristic Approach to Scheduling a Sales Summit. Selected papers of Proceedings of the 3rd International Conference on the Practice And Theory of Automated Timetabling, Springer LNCS vol. 2079, pp. 176-190, 2001.

[12] P. Cowling, G. Kendall and E. Soubeiga, Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation. Proceedings of the 2nd European Conference on EVOlutionary computation for Combinatorial OPtimisation. EvoCop 2002, Springer LNCS vol. 2279, pp. 1-10, 2002.

[13] M. Dorigo, Optimization, learning and natural algorithms, Ph.D. Thesis, Politecnico di Milano, Italy, 1992.

[14] M. Dorigo, V. Maniezzo and A. Colorni, The Ant System: Optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man and Cybernetics Part B, vol. 26, no. 1, 1996, pp. 1-13.

[15] M. Dorigo, V. Maniezzo and A. Colorni. Distributed optimization by ant colonies Proceedings of the first European Conference on Artificial Life Paris, ed. by FJ Varela and P Bourgine, MIT/Press/Bradford Books, Cambridge, Massachusetts: pp. 134-142, 1992.

[16] K. Dowsland, E. Soubeiga and E. Burke, Solving a shipper rationalisation problem with a simulated annealing hyperheuristic. Accepted for publication in the European Journal of Operational Research, 2005.

[17] A. Gaw, P. Rattadilok and R. Kwan, Distributed Choice Function Hyper-heuristics for Timetabling and Scheduling, Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT'04), pp. 495-497, 2004.

[18] L. Han and G. Kendall, An Adaptive Length Chromosome Hyperheuristic Genetic Algorithm for a Trainer Scheduling Problem, SEAL2002: pp. 267-271, 2002.

[19] L. Han and G. Kendall, Guided Operators for a Hyper-Heuristic Genetic Algorithm. In proceedings of AI-2003: Advances in Artificial Intelligence. The 16th Australian Conference on Artificial Intelligence (AI'03) (eds Tams D Gedeon and Lance Chun Che Fung), Perth, Australia, pp. 807-820, 3-5 Dec 2003.

[20] S. Petrovic and R. Qu, Case-Based Reasoning as a Heuristic Selector in a Hyper-Heuristic for Course Timetabling Problems, Proceedings of the 6th International Conference on Knowledge-Based Intelligent Information Engineering Systems and Applied Technologies (KES'02), vol. 82, Milan, Italy, pp. 336-340, Sep 16-18, 2002.

[21] S. Pimont and C. Solnon, A Generic Ant Algorithm for Solving Constraint Satisfaction Problems, Abstract Proceedings of ANTS' 2000 From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms, pp. 100-108, 2000.

[22] P. Ross, S. Schulenburg, J. G. Marin-Blzquez and E. Hart, Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02), pp. 942-948, 2002.

[23] P. Ross, S. Schulenburg, J. G. Marin-Blzquez and E. Hart, Learning a procedure that can solve hard bin-packing problems: a new GA-based approach to hyper-heuristics. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'03), pp. 1295-1306, 2003.

[24] K. Socha, J. Knowles and M. Sampels, A MAX-MIN Ant System for the University Course Timetabling Problem, Proceedings of the Third International Workshop on Ant Algorithms (ANTS'02), Springer LNCS vol. 2463, pp. 1-13, 2002.

[25] E. Soubeiga, Development and Application of Hyper-heuristics to Personnel Scheduling, Ph.D. thesis, University of Nottingham School of Computer Science, 2003.

[26] H. Terashima-Marin, P. Ross and M. Valenzuela-Rendn, Evolution of Constraint Satisfaction Strategies in Examination Timetabling, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99), vol. 1, pp. 635-642, 1999.