Ant Algorithm Hyperheuristic Approaches for Scheduling Problems

by Ross F. J. O'Brien, BSc (Hons)

Thesis submitted to The University of Nottingham
for the degree of Master of Philosophy, August 2007

Abstract

For decades, optimisation research has investigated methods to find optimal solutions to many problems in the fields of scheduling, timetabling and rostering. A family of abstract methods known as metaheuristics have been developed and applied to many of these problems, but their application to specific problems requires problem-specific coding and parameter adjusting to produce the best results for that problem. Such specialisation makes code difficult to adapt to new problem instances or new problems. One methodology that intended to increase the generality of state of the art algorithms is known as *hyperheuristics*.

Hyperheuristics are algorithms which construct algorithms: using "building block" heuristics, the higher-level algorithm chooses between heuristics to move around the solution space, learning how to use the heuristics to find better solutions. We introduce a new hyperheuristic based upon the well-known ant algorithm metaheuristic, and apply it towards several real-world problems without parameter tuning, producing results that are competitive with other hyperheuristic methods and established bespoke metaheuristic techniques.

Contents

List of Tables

List of Figures

Acknowledgements

> "If you're dumb, surround yourself with smart people. And if you're smart, surround yourself with smart people who disagree with you."
>
> – Isaac Jaffee, *Sports Night*

I'm not sure which category I fit into but I've managed to surround myself with a lot of smart people during my course, and I'd like to thank them here.

I would like to give thanks to the members, past and present, academic and administrative, of the Automated Scheduling, optimisAtion and Planning Research Group for their enduring support during the last five years, providing a rich multitude of voices and perspectives on life and work, always able to find something interesting to keep my own interest piqued and my fingers typing.

To the EPSRC for funding my research (under Grant number GR/N36387/01) during this time, and the University of Nottingham for assisting in that funding during the first couple of months.

To the members of the Document Engineering Laboratory Research Group for their comradery and encouragement.

To my parents and siblings for their love and patience.

To my best friends, for their tolerance, and for the bravery they've shown in the

journeys of their lives, reminding me of the potential I can yet achieve with mine.

To anyone who chooses to read this. I hope you find something of use.

Dedication

Dedicated to the memories of Frederick and Joan Jones, and Jim and Gladys O'Brien.

Declaration

I hereby declare that this thesis has not been submitted, either in the same or different form, to this or any other university for a degree.

signature:

Chapter 1

Introduction

"We have serious problems to solve, and we need serious people to solve them."

– President Andrew Shepherd, *The American President*

1.1 The Compromise of Optimisation Research

Much of combinatorial optimisation is compromise. Many real-world problems are not especially difficult to understand but they are sufficiently large and complex as to be computationally intractible (i.e. it would take an impractically large quantity of resources to find the globally optimal solution to a given problem formulation, which itself may be a compromise between criteria). A compromise is usually made to find as good a solution as possible with a reasonable amount of resources.

Considerable resources over the last half century or so have been devoted to the task of discovering and exploring methods which make use of problem-specific knowledge to reduce the solution space in order to make the search more manageable and efficient. A large number of simple heuristics and complex metaheuristics have been devised, presented and compared. While this is progress, there has been an ongoing concern that the field con-

centrates a great deal on benchmark problems and competitive advances, and has perhaps drifted away from real-world problems [Hoo96, McC06].

There is still a need for more general optimisation algorithms which can perform robustly on a set of problem instances or class of problems. Those who need such algorithms do not necessarily have the resources to sponsor researchers to spend years analysing their problems and fine-tuning algorithms which find good solutions to those problems in a reasonable period of time. A "rough and ready" approach is required: there is a need for more modular algorithms which can be "plugged in" and implemented with a minimum of fuss.

Hyperheuristics have relatively recently been focused upon as a more generic approach towards optimisation software. As opposed to methods which directly seek good solutions to individual problems, hyperheuristics work indirectly by managing the use of different direct methods to get the best from each.

In this thesis we investigate hyperheuristic approaches to several real-world problems: the Travelling Tournament Problem and a Project Presentation Scheduling Problem. Our intent is to reinforce the hypothesis that hyperheuristics are suitable for more general use.

1.2 Hyperheuristic Intent

As Ross [Ros05] phrased it,

> "The broad aim (of Hyper-heuristic) is to discover some algorithm for solving a whole range of problems that is fast, reasonably comprehensible, trustable in terms of quality and repeatability and with good worst-case behaviour across that range of problems."

A common 'slogan' associated with hyperheuristics is "good enough, fast enough, cheap enough" [BHK+03]. The intent is not simply to possess software capable of producing a solution of adequate quality in a reasonable period of time; its costs should also be low, meaning that much of the code can be reused easily and that what remains should be easy to develop or adapt for new problems. Transferability between problems means that the core framework, sometimes termed the hyperheuristic *black box*, should contain no problem-specific knowledge, and a hyperheuristic *interface* must exist to allow specific problems to relate to the framework. It naturally follows that an investigation into hyperheuristic methods must explore algorithms which use problem-generic information to make decisions.

It does not, however, naturally follow that we are seeking less than optimal solutions, or have lower expectations of solution quality; we aim for at least competitive results, qualified by the compromise that where a problem-specific algorithm may perform better on one given problem, or even one problem instance, and may not perform as well on another problem or instance (at least, not without significant – and costly – modification), the hyperheuristic algorithm will be readily applicable to both and produce competitive results for both.

One direction of the ongoing investigation into hyperheuristics has explored the use of well-known *metaheuristic* techniques, whose generic principles have generally been applied in problem-specific ways, as hyperheuristic techniques. This thesis presents a body of work exploring a hyperheuristic form of one such technique: the ant algorithm metaheuristic introduced by Dorigo [DMC96] and its contributions towards hyperheuristic learning.

1.3 Thesis Organisation

In Chapter Two we survey hyperheuristic literature, presenting the reasons for exploring generic optimisation methods as alternatives to specific methods and overviewing the ways in which the term "hyperheuristic" has been interpreted and applied. Chapter Three presents the black-box hyperheuristic framework, describing two real world test problems to which we will apply hyperheuristic methods, and describing the interface which allows hyperheuristic methods to perceive the problem in a domain-generic way.

In Chapter Four we present the Ant Algorithm Hyperheuristic, detailing its transliteration from metaheuristic to hyperheuristic, and testing it against several problems. In Chapter Five we conclude the thesis.

1.4 Contributions

The contributions of this thesis are:

- We introduce a new hyperheuristic technique to the literature, comparing its effectiveness against previously known hyperheuristics, and to some known metaheuristics in the literature.

- We demonstrate that hyperheuristics are capable of producing competitive results when applied to several real-world problems, without problem-specific parameter tuning.

- We present complete algorithms, with the full intent that future researchers may reproduce the results of this thesis and compare fairly.

1.5 Publications

As a result of the research reported in this thesis, the following paper has been published:

1.5.1 Conference Papers

E. Burke, G. Kendall, J. D. Landa Silva, R. O'Brien & E. Soubeiga, *An Ant Algorithm Hyperheuristic for the Project Presentation Scheduling Problem*, Proceedings of the IEEE 2005 Congress on Evolutionary Computation, Edinburgh, Scotland, Volume 3, pp. 2263-2270, September 2-5, 2005. [BKS+05]

Chapter 2

The State of the Art: Hyperheuristic Literature

> "Ultimately, of course, the only important question is, 'How do I find good solutions for my given cost function f?' The proper answer to this question is to start with the given f, determine certain salient features of it, and then construct a search algorithm, a, specifically tailored to match those features. The inverse procedure - far more popular in some communities - is to investigate how specific algorithms perform on different f's. This inverse procedure is *only* of interest to the degree that it helps us with our primary procedure, of going from (features concerning) f to an appropriate a."
>
> – D. Wolpert and W. G. MacReady, *No Free Lunch Theorems for Optimisation [WM97]*

In this chapter we survey the field of optimisation techniques, and hyperheuristic methods in particular. Our intent is to review the generality of approaches of optimisation software towards combinatorial optimisation problems, and present the need for more generic software, and the means by which hyperheuristics are currently trying to fulfil that need.

2.1 Heuristics

Many real-world situations involve the creation of schedules, including transport scheduling [WR95, Aro96], sports scheduling [ENT01, Ken07], school, course and examination

timetabling [BJKW97] and staff rostering [EJKS01], and research into the automation of creating good schedules has interested the academic community of Artifical Intelligence and Operations Research for decades [Wre95, Bow03].

Wren [Wre95], while deliberating the term *scheduling* and the many meanings it has acquired in the literature, considered the objective of scheduling to be:

> to solve practical problems relating to the allocation, subject to constraints, of resources to objects being placed in space-time, using or developing whatever tools may be appropriate. The problems will often relate to the satisfaction of certain objectives.

Such objectives often take the form of constraints which must be maintained or penalties which must be minimised. Constraints are often grouped into two types: *hard constraints* and *soft constraints*. Hard constraints cannot be violated under any circumstances, and solutions which satisfy these constraints are *feasible*. Soft constraints are desired but not essential, and mathematical formulations often assign weights to these soft constraints so as to quantify the cost of the solution. The optimal solution satisfies all hard constraints and minimises the accumulated penalty cost of all violations of soft constraints.

Optimisation problems are often complex, or have large (or infinite) *solution spaces*. The solution space of a problem is essentially the set of all solutions to that problem, irrespective of whether the solutions are complete or feasible, arranged in some complex topology which will be navigated by search algorithms. These issues of complexity and size make exhaustive searches impractical; exact methods such as linear programming methods and Lagrange multipliers are popular mathematical techniques, but they have difficulty scaling with NP-hard problems, which are characterised by the exponential increase in computational time needed to solve them to optimality as the size of the data increases.

Heuristic methods are often more practical, being derived from prior experience with the particular problem domain or otherwise specifically designed for it. They are therefore easily understood both by the developer of the optimisation software and those involved with the domain, which facilitates implementation, and they are capable of finding good solutions in a reasonable amount of time. This comes at the cost of certainty: heuristic methods do not (by definition) guarantee optimality or feasibility.

A heuristic method can often be described as a repeated use of one or more *heuristic operators*, often simply termed *heuristics*, which take an existing solution and modify it in some way. Heuristics are often grouped into two types: *constructive heuristics* which take an incomplete solution and typically choose the next element to schedule, producing a solution one element nearer to completeness, and *local-search heuristics* (also called *repair heuristics*) which take a complete solution and change it in some way, hoping to produce a solution which is more feasible or more optimal[1].

Heuristics are also often described by the means by which they modify the solution. The 'operator' aspect of the heuristic determines the nature of the modification, and the heuristic's *neighbourhood* is defined as being the set of 'neighbouring' solutions which can be reached within one operation: for example, the set of solutions with one more element scheduled, or the set of solutions which are only different by the swapping of two elements. Some heuristics are described as *greedy heuristics*, exploring all neighbours before selecting the neighbour which best satisfies the problem's objectives. Other heuristics use problem-specific information to pick a neighbour which best satisfies a particular objective or guideline or short-term need.

[1] respectively, has fewer hard or soft constraint violations

However, where a heuristic operator always seeks to find immediate improvement, the heuristic method is at risk of quickly finding a *local optimum*: a sub-optimal (and possibly infeasible or incomplete) solution which the heuristic operator cannot improve. In such cases it is often desirable to guide the heuristic search by employing a strategy known as a *metaheuristic*.

2.2 Metaheuristics

The term *metaheuristic* has come to be used in reference to a family of well-known search strategies. The essential qualities of each strategy are to encourage the discovery of better solutions in the search space by tightening a focus on "good" solutions and improving upon them (*intensification*), and to encourage the exploration of the solution space by broadening the focus of the search into new areas (*diversification*).

These two qualities are complementary, and necessary. A purely intensification-based search cannot accept poorer solutions and therefore cannot escape from local optima; a purely diversification-based search has no 'quality control' by which to reject poorer solutions and achieve good results.

We present a number of the better known metaheuristics. This is not intended to be a comprehensive review; the intent is to understand the principles behind these successfully applied techniques, so as to later explore the application of these principles to hyperheuristic techniques. In particular we present literature regarding the ant algorithm metaheuristic. Other reviews of metaheuristic literature can be found at [CDM+96, BK05].

2.2.1 Simulated Annealing

The *simulated annealing* metaheuristic was proposed by Kirkpatrick et al. in 1983 [KGV83]. The principle is to guide a search by automatically accepting better solutions (intensification) and probabilistically accepting poorer solutions (diversification), with the probability decreasing according to how poor the new solution is and how far the search has progressed. The technique is inspired by, and named after, the cooling schedules for annealed metals: early in the technique, with the temperature still high, the metals are very malleable and impurities can be more easily removed, whereas later the temperature is cooler and the metals are more resistant to change; by analogy, early in a search the simulated annealing metaheuristic allows a great deal of latitude in solution quality, hoping to remove the "impurities" from the solution before finally intensifying and accepting only improving moves, whereupon it converges. The cooling schedule, which determines the temperature/latitude at each decision point, may be static (i.e. pre-defined) or dynamic (adaptive, changing according to information gathered while the algorithm is run); such dynamics may include "re-heating", periodically increasing the probabilities of selecting poorer solutions at given times in order to diversify a search after a convergence. Since the method retains no memory of solutions or solution areas explored in its search, it has the advantage of being easy to implement. Similar metaheuristics include *Monte Carlo*, *Threshold Acceptance* and *Noising*: each accepts better solutions automatically, and poorer solutions according to some dynamic criterion.

2.2.2 Tabu search

Tabu search was proposed by Glover in 1977. The principle is to encourage a more diverse search by remembering solutions which have recently explored and excluding them ("making them tabu") from its current search. Considerable research had explored the scope of the "tabu list", i.e. the list of solutions or moves which are tabu at any time, in order to determine optimal ways of managing it. For example, a short list may be insufficient to prevent a cycling behaviour while a longer length costs more computational power (in terms of space, to store the tabu solutions or moves, and time to check the list) and may prevent the metaheuristic from moving to a good solution in order to find new parts of the solution space. The principle of tabu search has proven sufficiently useful that it is often hybridised with other metaheuristics. Glover and Laguna's book [GL97] is often cited as a key reference for users of tabu search.

2.2.3 Variable Neighbourhood Search

Variable Neighbourhood Search was proposed by Hansen and Mladenovic in 1995. To an extent the method is a precursor to hyperheuristic methods. The method proposes a search of a particular neighbourhood in hopes of finding a better solution. After sampling a neighbourhood for a time and not finding an improved solution, the method moves to the next neighbourhood in sequence and samples from it. If a better solution is found, the search begins anew from the better solution; this may simply involve returning to the first neighbourhood in the sequence, or it may re-order the heuristics according to identified characteristics of the solution. The nature and ordering(s) of the neighbourhoods

is manually defined, with each neighbourhood tending to have a broader scope or different neighbourhood to the previous neighbourhood. For example, a neighbourhood may explore all the ways in which a single event may be reassigned within a solution (1-opt). If the solution state is such that relocating a single event cannot produce an improvement, the method may move onto a second neighbourhood in which two events are moved (2-opt), perhaps swapping the events or otherwise moving one to make legal 'room' for the other. A survey can be found in [HM01].

2.2.4 Genetic Algorithms

Each of the metaheuristics described so far constructs or maintains a single solution, and may be considered *point-based* techniques. *Population-based* techniques construct or maintain many solutions.

Genetic algorithms are inspired by the natural processes of reproduction and evolution. The technique has been used since the late 1950s, though the seminal work is often attributed to Holland [Hol75]. At key decision points between cycles, termed as *generations*, poorer members of a population of solutions are removed and replaced by the "offspring" of the population, created by perturbing ("mutating") or breeding ("crossing over") the better members (usually as determined by their solution quality) in some way. The characteristic of genetic algorithms is its crossover operators, which in some fashion take components from two (or more) solutions and use them to create a third solution, theoretically taking good component parts of each solution and combining them to create a better whole; such operators must be specifically designed for the particular problem.

2.2.5 Ant Colony Optimisation

Ant Colony Optimisation (ACO) is the umbrella term for a number of techniques which have been developed during the last decade or so, beginning with the ant algorithm systems proposed by Colorni et al. [DMC96]. They are inspired by the real life means by which ant colonies find routes to food sources and communicate them to each other: they lay a substance called pheromone where they travel, which other ants can detect and follow. ACO techniques correspondingly employ a set of artificial agents (ants) to navigate a search space, laying a correspondingly artificial pheromone trail where they go and making their decisions based upon the pheromone trails left by other ants. The search space is usually represented by a network in which vertices represent assignments and decision points and arcs represent choices.

Ant colony techniques have been successfully applied to many problems. The standard technique, known as the ant algorithm technique, is usually 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.

An example of a non-standard ACO technique is presented in [CO98], as an application for the graph colouring problem [CO98]. A colony of ants is released upon an existing (and not necessary feasible) solution. Each ant traverses the graph, moving from vertex to vertex, making local-search repair efforts by identifying the most conflicted neigh-

bouring vertex and randomly choosing a new colour for that vertex such that it no longer conflicts. No pheromone is laid down, but the simple nature of each ant is consistent with ACO methods.

2.2.6 Asynchronous Teams

Asynchronous teams, also called A-Teams, were introduced in [TBGS98]. The team is a group of co-operating autonomous agents which operate upon a communal set of solutions which circulate around a network. Each agent's operator is free to transform a solution in any way, making it feasible or infeasible, complete or incomplete, and the solution is moved to a corresponding area of the network where further agents can transform them again. For example, events in a solution may be unscheduled and the the solution is moved to an 'incomplete' area where agents which take incomplete solutions as inputs may work upon them. Convergence occurs when a solution appears persistently; there is no overall co-ordination which determines which agents operate on each solution. 'Operations' in this context may be considered synonymous with 'heuristics'.

2.3 Generality in Methods

The cornucopia of established optimisation techniques naturally prompts the question for any new problem instance, 'which algorithm is the best for this problem?'. Indeed, there has been criticism of the competitiveness of the academic community in the ongoing presentation of algorithms whose major contribution appears only to be a slight improvement in objective satisfaction for some problem instance in a perhaps fractionally smaller period of computational time [Hoo96, SG06].

Such competitiveness was to some extent mollified by the publication of the No Free Lunch Theorem [WM97] by Wolpert and MacReady. The theorem states that all search algorithms have the same average performance when averaged over all problems defined on a finite search space. In other words, if an algorithm outperforms another on a given class of problems, another class of problem exists for which the algorithm is inferior: no individual algorithm is 'best'.

The authors cite as an example in [WM97] that generic methods (such as simulated annealing and the genetic algorithm) do not perform as well as specialised tailor-made methods for the Travelling Salesman Problem. The intent is to encourage the identification of salient features in the problem instance in order to properly match the instance with an appropriate algorithm, rather than to continue seeking an algorithmic 'holy grail' which is superior to all.

Several areas of operations research have developed over the years to tailor approaches to problem features, even before the No Free Lunch Theorem was published. Under human supervision, expert systems and knowledge-based systems have developed to reproduce the experienced manual approaches to particular problems such as educational timetabling. Case-based reasoning has developed more recently as a more adaptive framework for identifying the salient features for a given new instance of a problem class and matching the features with a particular algorithm, and deciding from the result of the match (the 'case') to reinforce or penalise its matching, and whether or not to remember the specific case to aid in identifying future problem instances. Interested readers may find [LKW97] useful.

However, such systems require an investment of interest and effort. A 'typical' set of problems must be considered in building and training the system's initial state, and such systems are designed in the knowledge that they will be expected to encounter new problem instances of the same class and adapt to incorporate the peculiarities of each new instance into the approach to the instances which follow it. The significant disadvantages of 'automated specialisation techniques' are their inapplicability to problems outside their specialised class, and their impracticality for 'one-off' problems.

The publications for the No Free Lunch Theorem advocate the identification of salient problem features in order to select the most appropriate algorithm, but the theorem does not discourage the use of generic techniques. Indeed, the theorem clearly indicates that if a specialised technique is better solving problems for its specific class than a generic method then problem classes must exist for which the generic method is superior – in fact, the specialisation of a method towards one problem class may guarantee its inferiority towards other problem classes simply because it is not practically applicable. A generic algorithm may not produce results which are superior to any algorithm tailored for its problem, but by being applicable to many different problems it may perform better on average than each tailored algorithm across the range.

2.4 Hyperheuristics

Hyperheuristic methods have been present in the literature since the early 1960s [FT61] but focus upon them has only really developed in the past decade or so. The term itself was coined in 2000 by Cowling et al. [CKS01a], to refer to a "heuristic which chooses

between heuristics", operating "at a level of abstraction above that of a metaheuristic", and proposed as an approach to increase "the level of generality at which optimisation systems can operate" [BHK+03], with which it is becoming synonymous.

The methodology was motivated by the awareness of many different approaches to different problems, each with different relative performances, particular advantages and individual flaws. In [BHK+03], Burke et al. reproduce an example (attributed to Ron Graham at AT&T Labs) of a one-dimensional bin packing problem for which a particular heuristic produces the optimal solution. The example presented is such that the removal of one particular object from the set to be packed, technically creating a smaller and intuitively easier instance, results in the heuristic counter-intuitively producing a worse solution.

In [LLP01], Lagoudakis et al. present a simple conundrum to illustrate the difficulties in selecting a particular algorithm to solve a problem. The conundrum is that of linear sorting, and choosing one of three well-known sorting algorithms based upon efficiency: insertion sort, merge sort, and quicksort. Quicksort is generally considered the fastest, owing to its better performance with increasing sizes of input, but its relative performance is poorer on smaller instances due to overheads of set-up. Lagoudakis exploit the recursivity of the merge sort and quicksort to create a hybrid technique which considers the size of the list to be sorted at each level and decide which to use to create two smaller instances, each upon which to choose another of the three sorting methods, with insertion sort, or a list of one element, being the terminal case. The hybrid, identifying 'ideal' input sizes for each algorithm, performs better than either of the three individual techniques did alone.

The principle of the hyperheuristic is similarly to manage a set of "low-level"

heuristics, each of which has its strengths and drawbacks when applied individually to a problem instance, to create an algorithm which is better than the sum of its parts. Cowling et al. describe it as managing "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" [CKS01a]. The further level of abstraction "above that of a metaheuristic" referred to by Cowling et al. is that of the "higher-level" heuristic, the hyperheuristic, being applied to the choice of heuristics rather than directly to the problem being solved: in a sense the hyperheuristic dynamically creates an algorithm to solve the problem, rather than solving it itself; as Hart et al. describe it, "Against the grain of most scheduling work, we are not interested in optimising any particular objective whilst producing the schedules, but in producing a fast and robust system that can produce high-quality practical schedules that satisfy a large number of constraints" [HRN98].

Overviews of hyperheuristics can be found at [BHK+03, Ros05].

We categorise various hyperheuristic approaches from the literature by their generality: the first category consist of *domain-knowledgable hyperheuristics* which are designed for specific problem classes, either mapping heuristics to solution states or evolving heuristic sequences for particular problem classes; the second category consists of *black box hyperheuristics* designed without any particular problem class in mind.

2.4.1 Domain-Knowledgeable Hyperheuristics

The category of domain-knowledgeable hyperheuristics could itself be divided into two further sub-categories. In one category, approaches are made involving specific problem instances. However, rather than directly create a solution for the instance, the algorithm

develops an 'approach': a solution constructor which combines heuristic information with solution state information. In the second category the "approach" is exclusively defined by heuristic steps to take and can be applied to any of a class of problem instances.

The earliest example of a technique resembling hyperheuristics is considered to have been published by Fisher and Thompson in 1961 [FT61]. They describe their algorithm for the job-shop scheduling problem as combining rules in a probabilistic learning way. Two simple rules for loading operations are presented. A control called *unbiased random process* picks either rule uniformly at random and applies the choice. The probabilistic learning method adapts its probabilities according to the performance of each rule, increasing the probability of selection if the rule performs well and decreasing the probability if it does not. The probabilistic learning system proves to be superior to the unbiased system, which in turn is superior to either of the individual rules.

In [FRC93, FRC94], Fang et al. develop an algorithm they term 'evolving heuristic choice'. Their intent is to improve the applicability of genetic algorithms, since effective crossover operators which ensure legal solution states are difficult to formulate and sometimes require complex encoding schemes to get the best from the genetic algorithm framework. They decide that a more flexible and intuitive approach, with respect to benchmark test job-shop scheduling problem and open-shop scheduling problem instances, is to develop chromosomes which encode sequences of (heuristic, operation) pairs, which instead of representing a solution, represents an ordering by which the operations are scheduled, each according to its respective constructive heuristic. The approach outperforms a number of solution-based genetic algorithms and in some instances (where the optimal solutions are

not known) finds new best solutions.

The problem Hart et al. approached in [HRN98] was of creating daily schedules for a Scottish chicken catching company. The approach divides the problem into two stages: in the first, orders are split into tasks to be assigned to squads of workers who can then catch the appropriate number of chickens for later transportation (the assignment sub-problem), and in the second each task is assigned a time interval such that the squads can co-ordinate with transportation vehicles and deliver their orders at the correct times (the timing sub-problem). A genetic algorithm handles these sub-problems simultaneously, using a chromosome made up of a permutation of orders and a sequence of splitting and assigning heuristics to apply to each order. The heuristics are based on domain-specific knowledge identified by examining the company's established practices, which created schedules by hand, in order to replicate the decision-making process of a human scheduler, and thereby create schedules which are more easily integratable with the company's established practices. Not all heuristics are valid when applied to particular orders; a repair mechanism replaces each invalid assignment heuristic with a random valid assignment heuristic, but where a timing heuristic produces an infeasible solution the chromosome is simply penalised. The genetic algorithm ultimately always found a feasible solution to each of its test cases.

Ross et al. continue their exploration of hyperheuristic techniques with the one-dimensional bin-packing problem, recognising the many heuristics identified for the well-known problem as an interesting demonstration of the potential of hyperheuristics. In [RSMBH02] a learning classifier system is used to classify different solution states and suggest a heuristic to place the next object into a bin. The heuristics are constructive

heuristics, drawn from previous literature: no new heuristics are introduced. The solution features to be identified with specific heuristics describe the proportion of remaining items of particular sizes, and the percentage of items remaining. The system is trained and tested on sets of benchmark problems from two operational research libraries. The resulting system performs no worse than the worst of the individual heuristics, and has a better overall performance than the best of them. The authors observe limitations of their method: the hyperheuristic does not take into account the current status of the already open bins, and provides rewards after every step, perhaps overemphasising the contributions of individual heuristics and not considering the collective contributions of several.

In [RSMBH03], Ross et al. develop a 'messy' genetic algorithm in which each chromosome in the population is a series of classifer blocks, each representing a set of circumstances in which to apply a particular heuristic. The results indicate a greater proportion of problem instances being solved to optimality but the emphasis of the results is on the reduced wastage of bins used by the algorithm: the genetic algorithm uses at most three more bins than the optimal reported solutions, and up to four bins less than any of the individual heuristics used.

In each of these cases the intent is to produce a system which uses a number of simple heuristics to offset any weakness any individual heuristic may possess. The heuristics are simple ones, based upon previous literature or previous manual experience with the problem domains and easily understandable to those who must use the resulting schedules (or oversee the hyperheuristic's performance), and therefore relatively easy to program and implement. However, while the heuristics themselves are relatively independent of the

system (as the chromosomes and classifers recognise them only as an encoded response for each chromosome block or classifer, the parameters of the solution state which trigger the heuristic via the chromosome or classifer must be identified prior to implementation and directly encoded, limiting the system's generality.

Hart and Ross also investigate a 'heuristically-guided genetic algorithm' as an approach to a job-shop scheduling problem [HR98]. In this case the chromosome directly represents a series of instructions to be applied in scheduling operations. The ith gene in the chromosome represents a (method, heuristic) pair in which at the ith decision point the corresponding method (from a choice of two) is employed to reduce the unscheduled events to a set of potential events to be scheduled next, and the heuristic chooses the one to be scheduled. The algorithm outperforms the genetic algorithm developed by Fang et al. [FRC93]. The authors also suggest a limitation of their own method that their sequences of heuristics depend heavily upon the performance of the first heuristics implemented, as these decisively determine the early shape of the schedule, and suggest that their method might be better employed in evolving a crucial partial schedule whose remainder might be solved by other means.

Rossi-Doria and Paechter employ a similar evolutionary algorithm to course timetabling problems, evolving sequences of pairs of heuristics which respectively choose a course to assign and then choose a timeslot to assign the course to [RDP03]. Their populations are seeded with a combination of single-heuristic chromosomes (which are considered to perform reasonably) and random mixtures, and the final sequences are reported as producing competitive results on two medium-sized instances and poorer results on the three others of

the set, with the discrepancy attributed to the possible absence of heuristics more appropriate for those instances from the heuristic sets. The heuristics used are based on well-known graph-colouring heuristics, with some of them weighting particular characteristics to better reflect the nature of the timetabling problem. The authors reflect that these heuristics are more computationally expensive than other possibilities since they must re-evaluate their graphs at every decision point.

In [TMFAR05], Terashima-Marin et al. extend the domain-specific classifier/GA hyperheuristic to two-dimensional cutting stock problems (which resembles a bin-packing problem, with bins substituted for standard-sized sheets). Terashima-Marin et al. develop the less specific genetic algorithm hyperheuristic, evolving chromosomes of pairs of heuristics [TMFZRVR06]. The first heuristic of each pair picks an object shape to be cut, and a sheet for it to be cut from, and the second heuristic determines the placement of the shape on the sheet. The two models were compared in [TMFZRVR07] using the two-dimensional bin-packing problem, with the 'pure' genetic algorithm appearing slightly better on average.

A concluding thought from this final example [TMFZRVR07] is that the result of each genetic algorithm is not simply a solution to a problem instance but an algorithm to approach any instance of that problem class whose average performance is hopefully better than any of its constituent parts (though individual heuristics may still perform better on particular instances). One of the common features of this problem class is that of size: each instance must have the same number of objects to assign since each heuristic pair in the chromosome encoding represents a single assignment. The genetic algorithm must therefore be rerun for problem classes of smaller or larger sizes, limiting the system's generality.

2.4.2 Black Box Hyperheuristics

This second category of hyperheuristic is defined by its indifference to problem structures, and its subsequent applicability to entirely different problem classes. In contrast to most domain-knowledgeable hyperheuristics, these mostly employ local-search/repair methods.

Nareyek employs a system for the Orc Quest problem and a Logistics Domain problem in [Nar03]. The system iteratively chooses a constraint to enforce and the constraint chooses from a range of problem-specific heuristics in order to reduce violations of the constraint. Reinforcement learning is employed to try and select the most appropriate heuristic at each decision point: each heuristic is assigned a utility weight which is rewarded and penalised as the heuristic respectively finds improving or non-improving solutions. The investigation considers many different combinations of reward/penalty schemes as well as whether to select the heuristic with the maximal weight or stochastically between heuristics according to their weights. General trends indicate that selecting the heuristic of maximal weight and providing small rewards and large penalties produces stronger results.

While Nareyek's reward system is based upon whether the heuristic simply produces an improvement or not, Cowling et al. [CKS01a] introduce a choice function hyperheuristic which uses the quantitative information of how much improvement of solution quality each heuristic produces in order to assign weighting values. The choice function is an aggregate of three values which evaluate each heuristic's relative merits. For the jth heuristic, the function will evaluate its recent *solo intensification* performance $f_1(j)$, its recent *sequential intensification* performance $f_2(i, j)$ (where i is a directly preceding heuristic move in the search) and a *diversification* factor $f_3(j)$ which reflects the CPU

time since the heuristic was previously used; the overall choice function is presented as $F(j) = f_1(j) + f_2(i, j) + f_3(j)$. Other than access to limited information about the evaluation function (i.e. what the current evaluation values are (for one or more objectives) and whether they should be maximised or minimised) and knowledge of how much CPU time has passed, the hyperheuristic possesses no domain specific information. The hyperheuristic is successfully applied to a sales summit problem and its autonomy is developed from a method with manually-tuned parameters to an autonomous method with adaptive parameters [CKS01b], with no training period or initial weights to manually pre-determine. In [CKS01b] it was observed that the quality of solutions produced was slightly disappointing in comparison to earlier tuned hyperheuristics, but that the solutions produced were still better than those produced by manual techniques, greedy heuristics, greedy and random hyperheuristics[2] and a simulated annealing metaheuristic developed for comparison purposes.

The choice function hyperheuristic's abilities are further demonstrated upon a project presentation scheduling problem and a nurse rostering problem [CKS02, KSC02, BKS03, Sou03]. The formulation and implementation of the approach to the presentation scheduling problem (beyond that of the pre-existing hyperheuristic framework) were measured at an equivalent of 101 man-hours, as an indication of how quickly the hyperheuristic method can be applied. The results for this problem were described as excellent compared to previous manual methods; the results of the hyperheuristic's application to the nurse rostering instances were shown to be competitive with an existing tabu search metaheuristic

[2]Probably the simplest of the random hyperheuristic class is the Simple Random Hyperheuristic, which chooses a heuristic at random with uniform probability at each decision point and applies it to the current solution. It is used in several hyperheuristic publications, including this thesis, [Sou03], and [AK03] as a control. A more detailed description of 'control' hyperheuristics can be found in Chapter 4, Section 4.2.

technique [Dow98].

Inspiration for further hyperheuristic techniques is derived from the metaheuristic family. Cowling et al. propose a genetic algorithm -based hyperheuristic (which they term 'HyperGA') in which chromosomes represent sequences of local-search heuristics [CKH02]. During each generation, chromosomes of heuristics are applied to the best known solution, and the best new solution of the generation becomes the starting point for the next generation. Chromosomes are evaluated according to the overall improvement generated by the sequence; variants divide this by the CPU time taken by the sequence to promote heuristics with shorter durations, and/or adapt the probability of mutation to the overall success of the generation. The intent is not to produce a specific sequence of heuristics which can produce good solutions (as is the case with various domain-knowledgable hyperheuristics) since the characteristics of the local solution space change from generation to generation, but to dynamically identify and reuse good sequences of heuristics to find better solutions. As applied to a trainer scheduling problem, the hyperheuristics perform better than any of the individual heuristics, and better than a genetic algorithm and memetic algorithm implemented for comparison purposes. Again, the parameter-tuned variant hyperheuristic performs slightly better than the adaptive variant. The fitness function variants which encourage the use of quicker heuristics perform less well. Though this is not commented upon in the paper, it seems possible that since the stopping condition of the experiment is a fixed number of generations (i.e. heuristic calls), the duration of any heuristic's implementation is not an issue and the performance of the hyperheuristic may be compromised by the emphasis of efficient heuristics over effective heuristics.

The HyperGA is experimented upon in further publications. In [HK02] Han and Kendall suggest that a fixed chromosome length is a possible limitation of the method, and propose an 'adaptive-length chromosome' hyperGA. Mutation operators are employed which identify good sub-sequences of heuristics in particular chromosomes to be inserted into other chromosomes, or identify poor sub-sequences of heuristics to be removed. In [HK03a] the chromosome incorporates a tabu value for each gene (heuristic): to improve the efficiency of the algorithm, heuristics which perform poorly are made tabu for the next several generations rather than being removed and possibly reinserted. Finally in [HK03b] the mutation operators provided by the adaptive-length chromosome hyper-GA are modified to guide the hyperheuristic towards good chromosome lengths, preventing chromosomes from becoming too short or too computationally expensive. In each case an improvement is generated over the previous best technique, and the best technique almost always uses fixed probabilities for crossover and mutations rather than adaptive values. The HyperGA's generality is demonstrated in [HK03b, Han05] as it is applied to the same project presentation scheduling problem as in [CKS02], the course timetabling problem (as approached in [BKS03], and some of whose instances are approached in [RDP03]) and an aircraft scheduling problem in [Han05].

Soubeiga initially considers simulated annealing as a comparison for his choice function hyperheuristic in [Sou03] when applied to the sales summit problem. The temperature is set at a fraction of the initial solution quality and follows a geometric path, decreasing each iteration by a fraction which is determined by prior experimentation. Except in the most difficult instances, the simulated annealing hyperheuristic produces better

results than the choice function for the real-world instances of the sales summit problem examined. This is attributed to its more frequent acceptance of poorer solutions early in the search, though the simulated annealing hyperheuristic makes all its heuristic choices uniformly at random. Bai applies a similar technique in [BK03, Bai05] to a shelf space allocation problem with a different temperature function. A number of monte-carlo acceptance criterions are explored by Ayob et al. [AK03] upon a route-planning problem involving the layout of printed circuit boards, with some better results than a choice function hyperheuristic. All temperature/monte-carlo functions appear to be tuned to their specific problems, exploiting details of the respective evaluation functions and heuristics (such as a known goal evaluation value of zero, or an known range of improvement values each heuristic can achieve in one application). In [DSB07] the formulation of a particular problem suggests that simulated annealing may be useful and this is used to guide the development of the hyperheuristic technique so as to make best use of the heuristics; while this undercuts the 'black-box' nature of the technique, the hyperheuristic framework allows for ease of implementation of the domain-knowledgeable components.

A number of different approaches have been described as 'tabu search hyperheuristics'. One approach is presented in [BKS03] by Burke et al.. In this system heuristics are applied and rewarded or penalised according to whether or not they produced an improvement, in a similar fashion to Nareyek, [Nar03]. During every cycle the highest ranked heuristic is applied. A heuristic which performs poorly is also added to a tabu list which prevents it from being applied until enough other heuristics have performed poorly enough that the hyperheuristic considers the new area of the solution space to be sufficiently different that

the heuristic might now be more appropriate. Since this mechanism takes control of tabu heuristics away from the learning aspects of the algorithm, it can no longer be considered simply a reinforcement learning technique. The paper demonstrates the hyperheuristics' performances on two different problems: the nurse rostering problem as approached in [BKS03] and the course timetabling problem approached by [Han05]. Bai considers this approach for bin packing problems in [Bai05].

A second 'tabu search hyperheuristic' approach is presented in [KH04a, KH04b] by Mohd Hussin and Kendall. At each decision point every available heuristic is implemented, with the best solution of those discovered being accepted as best. The heuristic which discovered the solution is added to a tabu list. This seems counterintuitive if one pursues the hypothesis that a good heuristic continues to be good in the near future, but the intent is to prevent the following cycle from exploring neighbourhoods which would return the search to already known solutions. The publications consider the size of the tabu list to be a parameter[3], though the application is only implemented upon a single problem class with a single set of heuristics, so no suggestions are made for adjusting the parameter to different problems. An extension is considered whereby the best heuristic of the cycle is repeated until it no longer produces an improvement. The hyperheuristic is tested upon a number of educational timetabling problems including benchmark instances and new instances encountered at a Malaysian University.

Cowling and Chakhlevitch consider several hyperheuristic approaches from a particular perspective: that a problem may have many potentially applicable heuristics and

[3] A tabu list of 0 naturally means every heuristic will be applied in every cycle, which constitutes a greedy hyperheuristic.

that the hyperheuristic should be able to manage a large number [CC03], where Soubeiga previously suggests in [Sou03] that a low number (around 10) is good. A number of basic hyperheuristics are considered with *random* (which always selects one heuristic uniformly at random) and *greedy* (which applies all, then selects the best) at the extremes. Several *peckish* hyperheuristics are described which maintain a candidate list of heuristics to be chosen from, and several tabu search heuristics are described which select good heuristics and make tabu any non-improving ones (and possibly a set of actual moves). The sizes of the candidate and tabu lists are adaptive in order to find a balance between intensification and diversification, and the peckish and tabu hyperheuristics deviate less from a given upper bound than the extreme hyperheuristics. In [CC05] further strategies are considered for determining the composition of the list of heuristics which can be considered during the cycle.

Burke et al. [BPQ06] and Petrovic and Qu [PQ02] consider a different approach to providing a manageable set of heuristics for the hyperheuristic. A case database is developed in which characteristics of timetabling problems are associated with an injection ratio for two graph-based heuristics (largest degree and saturation degree) to be applied in educational timetabling problems. The hyperheuristic creates a heuristic list in which solutions are constructed by iterating through the list and performing the corresponding heuristic; the injection ratio determines what portion of the heuristic list corresponds to the largest degree heuristic. The hyperheuristic runs the sequence of heuristics and then mutates the list in successive iterations. In [BDPQ05, QB08], Burke et al. and Qu and Burke explore hybridisations of their technique. A tabu list is constructed, containing heuristic

lists which lead to infeasible solutions. These lists are thus avoided by the list mutation operators. The variant algorithms also integrate local search during or after the solution construction. The case database was applied to randomly created and real-world problem instances and shown to produce better results than either of the individual heuristics alone or a previously well-considered probability weighting. Its results were consistently feasible (a quality not met by the individual heuristics) but were outperformed by a hybrid tabu search technique and state-of-the-art techniques. It was conjectured that with more sophisticated characteristic recognition and more learning time, the case-base system could yet perform better.

Many hyperheuristic publications focus on demonstrating the versatility and competitiveness of hyperheuristic methods with other methods in the literature, but most of these also propose new hyperheuristic techniques, in part to also explore the diversity of hyperheuristic techniques. Few studies exist with the intent of directly comparing hyperheuristic methods. Bilgin et al. [BOK06] initially consider a simple framework in which a hyperheuristic consists entirely of a heuristic selection mechanism (one of seven) and an acceptance criterion (one of five). Experimental results indicated that no particular combination was dominant over the others, though the acceptance criterion which accepted only solutions of equal or greater quality performed well and the Choice Function's performance was slightly better than other selection mechanisms[4].

Later, Ozcan et al. [OBK06] consider more complex frameworks recognising mutational and hill-climbing heuristics as distinct sets of heuristics. The intent is to explore

[4]No comment is made in the paper to the hyperheuristic framework illustrated in [Sou03], in which the Choice Function hyperheuristic is presented as a whole algorithm rather than a mere heuristic selection mechanism

whether or not the hyperheuristics benefit from being programmed with this knowledge and to gain a better understanding of the interactions of heuristic sets. The frameworks are applied to a set of benchmark functions representing a broad range of optimisation problems, with several of the new frameworks significantly outperforming the simple framework, and the paper concludes with the thought that a variety of heuristics, 'combined underneath a decent framework might generate a synergy, yielding a better performance'.

2.5 The Future of Hyperheuristics

The hyperheuristics described in this chapter so far, for all their differences, retain at least one common property: they are all designed to use only the heuristics with which they are provided, whether they are useful or not. Soubeiga demonstrated that hyperheuristics can adapt to the presence of problematic heuristics by introducing two extreme examples of poor heuristics to the Sales Summit Problem [Sou03], and Cowling et al. [CC03] demonstrated that an abundance of heuristics could still be managed, but very little work has been directed towards automating the process of creating new heuristics to cover any roles the provided set of heuristics do not fulfil.

Burke et al. [BHK06] have made a promising beginning in this direction by using Genetic Programming techniques to evolve heuristics for an online bin packing problem. Each evolved heuristic is evaluated in isolation, being iteratively presented with a new object, which it must place either into a new bin or a bin it has already placed objects into; in this online problem, these objects cannot be rearranged later and the heuristic has no knowledge of the objects it must yet place. The heuristic evaluates each existing bin in

turn, placing the object into the first bin for which the heuristic's evolved function exceeds a given threshold. The function is created from a limited number of terminal variables (the capacity and spare capacity of the bin and the size of the object) and simple operators (+, −, ×, %, <), and is solely responsible for the choice of bin: the system allows heuristics to place objects illegally into bins which do not have room for them, rather than forcing a choice between only those bins with spare capacity. This means that the quality of the final solution is entirely the result of the heuristic. As this final quality is used as a fitness function, heuristics which do not overfill bins are more likely to be kept into successive generations. It was observed that simple heuristics, i.e. those with small function trees, performed very well, and this prevented code-bloat from becoming a significant issue; the system succeeded in evolving the first-fit heuristic.

Further work in this area has developed towards identifying qualities about a given problem instance and providing a heuristic to match [BHKW07a], and expanding the scope of the algorithm to more complex problems: a 2D bin-packing problem [BHKW07b]. It remains to be seen whether genetic programming can be employed in a hyperheuristic framework to create and adapt a set of heuristics to the problem instance they are trying to solve. Until then it seems that hyperheuristics must continue to manage human-designed heuristics.

2.6 Conclusion

We have reviewed some of the principles of existing and developing heuristic, metaheuristic and hyperheuristic techniques in the literature, and identified reasons why problem-generic

software is useful and gaining popularity. In the next chapter we examine the black-box hyperheuristic as it is applied to two problem classes.

Chapter 3

Hyperheuristic Framework and Optimisation Problems

3.1 Introduction

The black box hyperheuristic framework is often diagrammatically represented in a form resembling Figure 3.1 [Bai05]. A problem class, together with its solution representations, low-level heuristics and evaluation functions, constitutes the domain-specific side of the framework and this must be implemented afresh for each new problem class. A hyperheuristic, which regards solutions, heuristics and evaluation functions as more abstract concepts, is an entire module which may be re-used without modification for each new problem class. An interface connects the two sides, ensuring that the hyperheuristic has access to all the information it can be reasonably expected to use.

In this chapter we expand upon this diagram. Two real-world optimisation problems, both already described in published literature, are presented as the hyperheuristic framework will perceive them. The interface is described to aid the understanding of how

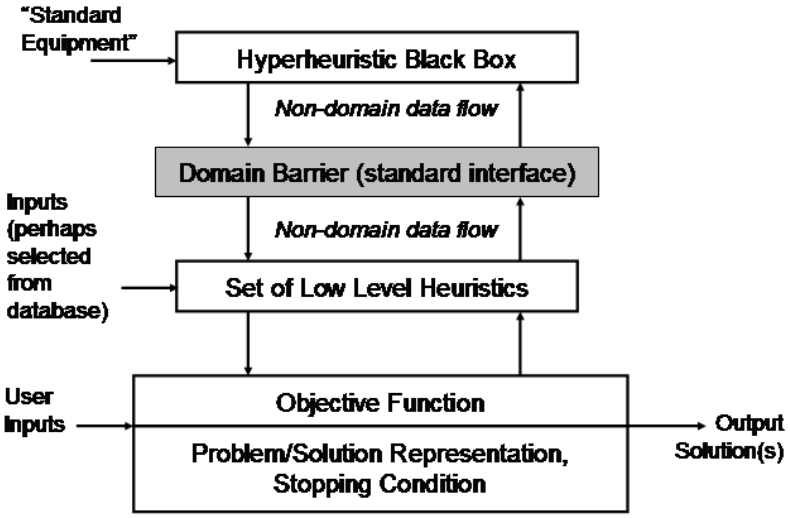

FIGURE 3.1: The Hyperheuristic Framework

further problems can be similarly represented for the framework. Finally we remark on the experiments we will be performing upon these two problems.

3.2 Project Presentation Scheduling Problem (PPSP)

3.2.1 Introduction and Formulation

The Project Presentation Scheduling Problem is a small real-world problem introduced in 2002 [CKS02]. The formulation and instances we use are specific to the School of Computer Science at the University of Nottingham, though no doubt variations on this problem exist elsewhere in other academic facilities, and other real world applications such as conference presentation scheduling use the same principles.

Final year students undertaking the Computer Science undergraduate degree at the University of Nottingham are required to complete a year-long project under supervision from a member of staff. One aspect of their assessment is an individual fifteen-minute presentation (including five minutes for questions) given before at least three members of academic staff, who are identified as the First marker, Second marker and Observer. It is preferred that the student's supervisor is among the three markers and that the markers all have interests in the subjects of each presentation.

For the purposes of the scheduling, presentations are grouped into hour-long *sessions* which correspond to an hour of the working week. Students are assumed to be able to present at any session, though individual members of staff or rooms may be unavailable for particular sessions; some sessions are also identified as *bad* (for example the 9-10am and 5-6pm sessions). The mathematical formulation of the problem seeks to fairly distribute the numbers of presentations, sessions and bad sessions that each staff member is assigned to attend. A survey is made of the subject of each presentation to determine which fields of Computer Science the presentation might appeal to, and a similar survey is made of the staff members to determine which fields each is interested in, to assist the matching of staff members to interesting presentations.

A solution to the problem is represented as a series of presentation assignments: each presentation is assigned a session number, a room, and three staff members. This allows some flexibility, in that potentially more than three members of staff may be assigned to a particular session/room pair. The solution and heuristics maintain feasibility by ensuring that no more than four presentations can be assigned to a particular session/room pair and

that no staff member is assigned to presentations in more than one room during a session in which they are available, or to any presentations in a session they are unavailable for.

We use three instances of the problem, labelled *csit0*, *csit1* and *csit2*. Instance *csit0* (from the 2000-2001 academic year) seeks to schedule 151 presentations and 26 staff members between 80 sessions and 2 rooms; instance *csit1* (from the 2001-2002 academic year) seeks to schedule 240 presentations and 24 staff members between 36 sessions and 2 rooms. Instance *csit2* is almost identical to *csit1*, differing only to declare two staff members absent during the presentation timetable; they are therefore unable to mark the presentations of students they have supervised, and the fair distribution of assessment to staff members is unbalanced.

The problem instances are considered to progress in increasing difficulty as the constraints tighten in later instances, with more presentations to schedule into less sessions. The optimal solution(s) for each instance have not been provably identified.

The problem is formulated as a minimisation problem, as follows:

$\mathbf{I}$ is the set of students
$\mathbf{S}$ is the set of staff members
$\mathbf{Q}$ is the set of sessions, with subset $\mathbf{Q_{bad}}$ being the set of bad sessions
$\mathbf{R}$ is the set of rooms

x_{ijklqr} is an assignment of value $\begin{cases} 1 & \text{if student } i \text{ presents at session } q \text{ in room } r \text{ before First Marker } j, \text{ Second Marker } k \text{ and Observer } l \\ 0 & \text{otherwise} \end{cases}$

where $i \in \mathbf{I}$, $j, k, l \in \mathbf{S}$, $q \in \mathbf{Q}$, $r \in \mathbf{R}$, $j \neq k$, $j \neq l$, $k \neq l$

y_{jqr} is an assignment of value $\begin{cases} 1 & \text{if staff member } j \text{ is in attendance at session } q \text{ in room } r \\ 0 & \text{otherwise} \end{cases}$

where $j \in \mathbf{S}$, $q \in \mathbf{Q}$, $r \in \mathbf{R}$

p_{ij} is the number of common interests between presentation i and staff member j,

and $Sup_{ij} = \begin{cases} 1 & \text{if staff member } j \text{ is the supervisor of project } i \\ 0 & \text{otherwise} \end{cases}$ where $i \in \mathbf{I}$, $j \in \mathbf{S}$

The objective is to:

Minimise $E_P = A + 0.5B + 0.3C - D$

such that

$\Sigma_{j,k,l \in \mathbf{S}} \Sigma_{q \in \mathbf{Q}} \Sigma_{r \in \mathbf{R}} x_{ijklqr} = 1$, $(i \in \mathbf{I})$
$\Sigma_{i \in \mathbf{I}}, \Sigma_{j,k,l \in \mathbf{S}} x_{ijklqr} \leq 4$, $(q \in \mathbf{Q}, r \in \mathbf{R})$
$\Sigma_{r \in \mathbf{R}} y_{jqr} \leq 1$, $(j \in \mathbf{S}, q \in \mathbf{Q})$
$x_{ijklqr} \in 0, 1$, $i \in \mathbf{I}$, j,k,$l \in \mathbf{S}$, $j \neq k$, $j \neq l$, $k \neq l$, $q \in \mathbf{Q}$, $r \in \mathbf{R}$

A represents the fair distribution of the total number of presentations per staff member, B represents the fair distribution of the total number of sessions per staff member, C represents the fair distribution of "bad" sessions per staff member and D represents the level of interest the assigned staff members have in the presentation, i.e.

$$A = \Sigma_{j \in \mathbf{S}} \left(\Sigma_{q \in \mathbf{Q}} \Sigma_{r \in \mathbf{R}} \Sigma_{i \in \mathbf{I}} \Sigma_{k,l \in \mathbf{S}} (x_{ijklqr} + x_{ikjlqr} + x_{ikljqr}) - \frac{3|\mathbf{I}|}{|\mathbf{S}|} \right)^2$$
$$B = \Sigma_{j \in \mathbf{S}} \left(\Sigma_{q \in \mathbf{Q}} \Sigma_{r \in \mathbf{R}} y_{jqr} - \frac{4|\mathbf{Q}|}{|\mathbf{S}|} \right)^2$$
$$C = \Sigma_{j \in \mathbf{S}} \left(\Sigma_{q \in \mathbf{Q}_{bad}} \Sigma_{r \in \mathbf{R}} y_{jqr} - \frac{4|\mathbf{Q_{bad}}|}{|\mathbf{S}|} \right)^2$$
$$D = \Sigma_{j \in \mathbf{S}} \left(\Sigma_{q \in \mathbf{Q}} \Sigma_{r \in \mathbf{R}} \Sigma_{i \in \mathbf{I}} \Sigma_{k,l \in \mathbf{S}} (p_{ij} + 10 Sup_{ij})(x_{ijklqr} + x_{ikjlqr} + x_{ikljqr}) \right)$$

where $\mathbf{Q}$ (respectively $\mathbf{Q_{bad}}$) is the set of sessions (bad sessions) used in the schedule. $\frac{3|\mathbf{I}|}{|\mathbf{S}|}$, $\frac{4|\mathbf{Q}|}{|\mathbf{S}|}$ and $\frac{4|\mathbf{Q_{bad}}|}{|\mathbf{S}|}$ respectively are the fair number of presentations, sessions and bad sessions each staff member should attend. We note that since staff members are assigned to presentations, not sessions, it is possible for more than three staff members to attend a session, though only three at any time count as markers. This occasional redundancy means that the average number of sessions per staff member is sometimes more than the fair number; better solutions resolve the redundancy.

3.2.2 Other Approaches

This particular problem was originally solved manually by a course convener. The objectives in this manual process emphasised a fair distribution of staff members across presentations, sessions and bad sessions, but did not take preferences for project topics into account. The intent of Cowling et al. [CKS02] in automating the process was to demonstrate that hyperheuristic techniques could be quickly applied to new problems with minimal development time. The problem was modelled, incorporating project subject preferences, in collaboration with the course conveners. They recorded the development of this problem formulation, a constructive heuristic to produce good initial solutions and a set of low-level heuristics to interface with their existing hyperheuristic framework at 101 hours, equivalent to two-and-a-half ordinary working weeks. They set a running time of ten minutes for the hyperheuristic to develop its initial solution. The course convener described the results of the algorithm as "excellent"; publications by the authors include a much poorer evaluation value from a solution created by the manual process, but do not elaborate on that process or how long it took. A solution produced by the hyperheuristic was put into practice, and one of the authors confirms the schedule ran smoothly [Sou03], and that the hyperheuristic was used again the following year to schedule presentations.

The problem formulation, the constructive heuristic used to create the initial feasible solution, and the eight local-search heuristics presented here were all initially presented in [CKS02]. The Choice Function and several simple hyperheuristics were presented and applied to the problem, with the Choice Function producing the best results. Further approaches to this problem were explored in [KSC02, Sou03], in which it was determined

that a Choice Function derived from the compound objective function (described above) performed better than an amalgamation of Choice Functions derived from the individual objectives. The problem was also used to demonstrate the versatility of Han and Kendall's Genetic Algorithm -based Hyperheuristic, the Hyper-GA [HK03b], and the most recent variation on that technique, the Tabu-assisted Adaptive-Length-Chromosome hyper-GA with guided operators, produced improvements upon those results.

To date these specific instances have only been attempted by hyperheuristic methods.

Approach	*csit0*	*csit1*	*csit2*	Time
Manual Process	-90.1			
Soubeiga, Random HH [Sou03]	-1193.19	-2880.93	-1614.11	600
Soubeiga, Choice Function HH [Sou03]	-1444.99	-2963.37	-1724.15	600
Han and Kendall, HyperGA [HK03b]	-1453.32			924

TABLE 3.1: Averaged results of approaches to the Project Presentation Scheduling Problem.

3.2.3 Hyperheuristic Approach

We use the same constructive heuristic to create an initial feasible solution as Soubeiga [Sou03]; this heuristic is non-deterministic. Unfortunately this also means that we do not have access to the same initial solution(s); for comparative purposes we ran the algorithm twenty times, and used the solution whose evaluation value was closest to the published initial evaluation value to initialise our hyperheuristics. Nevertheless, we present the algorithm:

- This algorithm iterates through a number of nested loops, assigning presentations until none remain.
- The outermost loop iterates through the available staff members, selecting the first staff member at random.

- The second loop iterates through sessions which the staff member is available for (there may be particular timeslots which the staff member is unavailable for). The loop first iterates through sessions which are not bad, then sessions which are bad.
- The third loop iterates through available rooms. A room may be unavailable during a given session, or the first staff member may already be assigned to presentations during that session in a different room, or it may not be possible to assign any more presentations to that session for that room.
- The fourth and fifth loops then select two further distinct staff members who are available for presentations during that session in that room. There is no effort made, at this stage or in the outermost loop, to optimise the 'fairness' objectives of the evaluation function by preferring staff members with fewer assigned presentations/sessions/bad sessions.
- Within this innermost fifth loop, the algorithm evaluates unscheduled presentations, preferring the presentations of any students whose supervisor is one of the three staff members. If there are any such presentations, the rest of the list is ignored. The list is then reduced to a target size, equivalent to the number of presentations which could be scheduled to a session, by selecting those with the highest match between presentations and staff interests.
- The reduced list of presentations are then scheduled. If a student's supervisor is one of the three staff members, then the supervisor is automatically the Observer of that presentation; the staff members otherwise respectively take on the roles of First marker, Second marker, and Observer.

We also use the same eight non-deterministic low-level heuristics:

Ph_1 Replace a random lecturer j_1 in a random session in which he/she is scheduled for presentations with a second random lecturer j_2.

Ph_2 The same as Ph_1 but j_1 has the largest number of scheduled sessions.

Ph_3 The same as Ph_2 but the session chosen is the one where j_1 has the smallest number of presentations.

Ph_4 Move a random presentation i from its current room-session assignment to another.

Ph_5 The same as Ph_4 but presentation i is that for which the sum of presentations involving all three involved lecturers is smallest of all sessions.

Ph_6 The same as Ph_5 but the new session is one in which at least one of the involved lecturers is already scheduled to mark presentations.

Ph_7 Swap the 2nd marker of one random presentation with the observer of a random other (a supervisor may not be removed).

Ph_8 Swap the 1st marker of one random presentation with the 2nd marker of a random other (a supervisor may not be removed).

We observe that heuristics Ph_1 and Ph_4 intuitively prove that the entire solution space is available to the hyperheuristic.

3.3 Travelling Tournament Problem (TTP)

3.3.1 Introduction and Formulation

The Travelling Tournament Problem is a relatively new real-world problem introduced by Easton, Nemhauser and Trick [ENT01] in 2001. Inspired by professional sports tournaments, specifically the American NFL baseball league, the task is to generate a double round-robin tournament schedule for a given number of teams n (where n is even).

Each team has a *home venue*, and is required to play against every other team twice during the schedule, once at their own home venue (known as a *home match*), and once at their opponent's home venue (known as an *away match*). During each timeslot, each team will play exactly one match; therefore there are $\frac{n}{2}$ matches per timeslot and the schedule takes $2n - 2$ timeslots to complete.

Each team's personal schedule consists of a permutation of $n - 1$ home matches

and $n-1$ away matches. Teams travel between timeslots, from the venue where one match is held directly to the venue where the next match is held. The teams always begin the schedule at their own home venues and always return there at the end of the schedule[1]. Consecutive home matches, during which a team remains at their home venue and does not travel, are known as *home stands*. Consecutive away matches, during which a team travels directly from opponent's venue to opponent's venue between matches, are known as *road trips*. A matrix of size $n \cdot n$ provides the distances between each pair of venues.

Hard constraints are placed on the lengths of home stands and road trips. Typically, a home stand or road trip cannot consist of fewer than one match or more than three matches. Each occurrence of a stand or trip consisting of fewer or more matches than permitted constitutes a hard constraint violation in the schedule. In addition, a team is not permitted to play against any opponent twice in consecutive timeslots (alternating home and away venues): such occurrences are known as *repeaters* and also constitute a violation.

The objective of the problem is thus to produce a schedule in which all matches are played, no hard constraints are violated and in which the total distance travelled by the teams is minimised.

The inputs are therefore: n, the number of teams; D, an $n \cdot n$ distance matrix; L, U, the lower and upper boundaries for the lengths of home stands and road trips.

The problem is formulated as a minimisation problem. The objective is to:

$$\text{Minimise } E_T = ((H \cdot W) + 1) \cdot \sum_{t=1}^{n} TD_t$$

where H is the number of hard constraint violations in the schedule, W is the weight of a

[1]Easton et al. observe in [ENT02] that the logical extremes of such a schedule, for each team, are an arbitrary alternation of home and away matches or a long stretch of home matches followed by a Travelling Salesman Problem -route through all away venues.

TABLE 3.2: The *NL6* Dataset

Match Venue	Distance to Match Venue					
	1	2	3	4	5	6
1	0	745	665	929	605	521
2	745	0	80	337	1090	315
3	665	80	0	380	1020	257
4	929	337	380	0	1380	408
5	605	1090	1020	1380	0	1010
6	521	315	257	408	1010	0

hard constraint violation relative to one unit of distance ($W = 100$ is common) and TD_t is the total distance travelled by team t during the schedule.

Sixteen instances of the problem exist. The real-world sports league which inspired the formulation of the TTP had sixteen teams, and the first seven problem instances were generated in 2001 by progressively removing fewer teams from this league; these instances are labelled for the number of teams involved: *NL4*, *NL6*, *NL8*, *NL10*, *NL12*, *NL14* and *NL16*. In 2006, following an increase in interest in the problem, nine further instances were generated, including a second instance with 16 teams (*NL16-2*) and eight instances with progressively more teams added: *NL18*, *NL20*, *NL22*, *NL24*, *NL26*, *NL28*, *NL30* and *NL32*. The distance matrix of instance *NL6* appears in Table 3.2.

Instances with four teams (such as *NL4*) are considered trivial (since only two matches can occur each timeslot, and there are thus only $6! \times 2$ solutions to consider). Instance *NL6* has been solved to optimality, but is still considered a difficult problem; the other instances have not been solved.

An example solution for instance *NL6* appears in Table 3.3. For each match, the home team is given by the row number and their opponent by the column number; the

Home	Away Team / Timeslot						Timeslot / Match Venue										Distance
Team	1	2	3	4	5	6	1	2	3	4	5	6	7	8	9	10	Travelled
1	-	8	3	4	7	5	3	2	1	1	1	5	1	1	6	4	4558
2	2	-	6	1	9	4	2	2	4	2	5	2	3	1	2	6	4974
3	1	7	-	9	10	2	3	3	1	5	4	2	3	6	3	3	3581
4	10	3	5	-	8	7	2	5	4	1	4	6	4	4	3	4	6241
5	6	5	4	2	-	1	5	5	6	5	5	5	1	4	2	3	4991
6	9	10	8	6	3	-	5	3	6	2	1	6	4	6	6	6	4684
																Total Distance Travelled	29029

TABLE 3.3: An *NL6* solution. In the first section of the table, each team (rows) plays its home matches against the away teams (columns) in the specified timeslots. In the second section, this corresponds to a schedule where each team will be located at the specified venue at each timeslot (columns), thus each team is at home when playing home matches and at other venues when playing away. In the third section, each team's schedule has a corresponding distance travelled.

corresponding table value is the number of the timeslot in which the match occurs. Thus team 1 plays at home against team 2 in timeslot 8. It can be seen that teams 1 and 5 play at team 5's home venue in timeslot 6 and team 1's home venue in timeslot 7, thus constituting a repeater - in fact, timeslots 6 and 7 comprise three repeaters, involving all six teams. It can also be seen that team 3 has four consecutive away matches in timeslots 3-6 (playing against, in order, teams 1, 5, 4, and 2) and team 5 has four consecutive away matches in timeslots 7-10 (against teams 1, 4, 2 and 3), thus breaching the upper boundary of the road trip constraint. There are thus 5 hard constraint violations.

The final column of the table is the distance travelled by each team. The total distance travelled is thus 29029 units.

The fitness of the initial solution in Table 3.3 is

$$E_T = ((H \cdot 100) + 1) \cdot \sum_{t=1}^{n} TD_t = ((5 \cdot 100) + 1) \cdot 29029 = 14543529$$

3.3.2 Other Approaches

The Travelling Tournament Problem was presented in [ENT01]. Easton et al. comment on the applicability of exact methods, commenting that the Problem's feasibility issues suggest constraint satisfaction techniques as interesting while its optimisation techniques suggest integer programming techniques, but that both approaches (summarised in the paper) have difficulty with it.

Easton et al. continue in [ENT02] to present an integer programming and constraint programming hybrid approach. The approach begins with an analysis of each team's individual tour possibilities, producing a subset of optimal team tours which are branched-and-priced and recombined.

Anagnostopoulos et al. successfully applied a simulated annealing approach to the problem [AMHV03], producing for a long while some of the most optimal solutions to several problem instances (*NL8*, *NL12*, *NL14* and *NL16*). The approach utilises five neighbourhoods to explore the search space (re-used in the approach presented here), and reheats and objective function adjustments to reinvigorate the search. More recently Van Hentenryck et al. have improved further on these solutions [Tri07].

Di Gaspero and Schaerf later applied a tabu search approach to the problem [GS05]. An initial solution is created by randomly assigning teams and timeslots to a tournament pattern generated from a so-called canonical pattern. Two recover-chain neighbourhoods, borrowed from Anagnostopoulos et al., are used to explore the search space and the tabu list takes into account all possible combinations of random values which result in the same neighbourhood move.

A hyperheuristic method based upon the ant algorithm hyperheuristic presented in this thesis (as described in [BKO+03, BKS+05]) has also been successfully applied to the problem [CKB07, Che07].

Approach	*NL6*	*NL8*	*NL10*	*NL12*	*NL14*	*NL16*
Cardemil		40416	66037	118955	205894	308413
Dorrepaal			66369	119990	189759	
Easton	**23916**	**39721**				312623
Langford			**59436**	112298	190056	272902
Lanchi, Lapierre and Laporte				138850	262010	
Rottembourg and Laburthe			68691	143655	301113	
Shen						281660
Trick	23978					
Van Hentenryck			59583	**110729**	**188728**	**261687**
Zhang		39947	61608	119102	207075	293175
Chen	**23916**	40391	65168	123752	225169	321037

TABLE 3.4: Best results of approaches to the Travelling Tournament Problem, data provided at [Tri07], with results from hyperheuristic approach by Chen provided at [Che07]. *"Van Hentenryck" is Anagnostopoulos, Michel, Van Hentenryck and Vergados*

3.3.3 Hyperheuristic Approach

We represent a solution to the problem as an assignment of matches to timeslots. We are able to guarantee partial feasibility in that we ensure all matches are always scheduled in such a way that every team plays exactly one match per timeslot, but the absence of repeaters, home stands and road trips is not guaranteed. A solution is considered fully feasible if no hard constraints are violated.

We use a simple recursive algorithm to construct a partially feasible tournament pattern (to borrow Di Gaspero and Schaerf's term [GS05]) for the appropriate number of teams, presented as follows:

- This algorithm creates a tournament pattern P_n for n teams (remember, n is even) by

recursively producing a tournament pattern P'_n for $\frac{n}{2}$ teams. At each level the teams are halved and a mirrored schedule is created for both halves, and the remaining timeslots of the schedule are created by rotating each team from one half with each team from the other. Where $\frac{n}{2}$ is odd, we insert a "blank team" into each half, creating a schedule for $\frac{n}{2}+1$ teams, and recombine them to remove the blanks.

- If $n = 2$, let the two teams play during the first timeslot. This is the terminal schedule.
- If $n \neq 2$, create a pattern P'_n for $\frac{n}{2}$ or $\frac{n}{2}+1$ teams, as described here.
- Mirror pattern P'_n, such that for any timeslot t and any pair of teams i and j playing each other in pattern P'_n, teams $i+\frac{n}{2}$ and $j+\frac{n}{2}$ also play each other at timeslot t.
- If $\frac{n}{2}$ is odd, remove the blanks from the mirrored half-schedules. This is done by identifying, for all timeslots, the two teams scheduled to play against blank teams, and pairing them against each other.
- Fill in the rest of the schedule by creating a rotation for each timeslot: In the first timeslot, have team i play team $i+\frac{n}{2}+1$; in the second timeslot, have team i play team $i+\frac{n}{2}+2$ and so on. If $\frac{n}{2}$ is even, the final week will be made up of matches where team i plays team $i+\frac{n}{2}$; if $\frac{n}{2}$ is odd, these matches will already be scheduled where teams initially scheduled to play blank teams have been scheduled against each other.

This algorithm creates a schedule in which every team plays every other team exactly once. A random permutation of the teams is created and assigned, such that teams earlier in the ordering will be assigned as the home team in any match they must play against a team later in the ordering, and the timetable is then mirrored for the corresponding away matches. The timeslots are similarly permutated.

We use the same five low-level heuristics as Aganostopoulos et al. [AMHV03] in our hyperheuristic framework. All five heuristics are non-deteministic.

Th_1 selects two random teams, T_i and T_j, and swaps the scheduling of the two matches in which these teams play, i.e. the operator finds the match in which T_i would play at home against T_j and has T_j play at home instead, and similarly with the match in which T_j had played at home against T_i.

Th_2 selects two timeslots at random and swaps their matches over.

Th_3 selects two teams at random and swaps their schedules such that where T_i was assigned to play a given opponent at a given venue in a given timeslot, T_j must now do so, and vice versa (except during those timeslots in which the two teams would play each other).

Th_4 selects two timeslots at random, as in h_2, and moves one random match from one timeslot to the other. A recovery chain of swaps occur to restore partial feasibility. At the extremes, this may mean that a match is swapped with its repeater, duplicating the effect of Th_1, or that every match between the timeslots is swapped, duplicating the effect of Th_2.

Th_5 selects two teams at random and swaps part of their schedules across, with a recovery chain effect similar to Th_4, and logical extremes of Th_1 and Th_3.

Di Gaspero and Schaerf used only heuristics Th_4 and Th_5 in their approach. Chen's hyperheuristic approach [Che07] adds five greedy heuristics, corresponding to these five, which consider every combination of timeslots and teams before choosing the best move. We can infer that heuristics Th_1 and Th_2 must be able to explore all possible solutions to instances involving four teams (such as *NL4*), but as such instances are trivial we do not explore them here. However, we note that it is not intuitively apparent from these heuristics that the entire solution space of the larger instances is available to the hyperheuristic.

3.4 Interface

The generic module of the hyperheuristic framework is termed the *learning mechanism* by Soubeiga [Sou03], though several of the hyperheuristics described in that work involve no

learning. This module is aware of solutions, evaluation functions and heuristics, but only in a limited sense; solutions are objects; evaluation functions are means to determine whether one solution is "better" (more complete, more feasible, more optimal) than another, but the hyperheuristic has no means of analysing the evaluation function to determine how much better; heuristics can be applied to solutions, discovering new solutions. The generic module decides which of its available heuristics should be applied to any of its known solutions at any given decision point.

We can express this process as a cycle, represented in Figure 3.2. A hyperheuristic is provided with inputs including a set of initial solutions and must produce an output set of solutions. A decision is made iteratively to either end the search or continue it; this decision may be made based on the hyperheuristic achieving a certain level of solution quality (e.g. optimality, feasibility) or on some pre-specified stopping condition (e.g. a given period of time, a given number of heuristic calls, a given duration in which no absolute improvement is made).

The decision phase of the cycle (the aforementioned *decision point*) is the phase in which a combination of heuristics are selected and solutions are selected for those heuristics to be applied to. It comprises both the processes of making the new choice and the learning from previous choices.

The application phase of the cycle is outside the hyperheuristic's control; it is the phase of interaction between heuristics and the solution space which discovers new solutions and new information about each heuristic to be assimilated during the next decision point. These interactions are domain-dependent; sections 3.2 and 3.3 describe two domains and

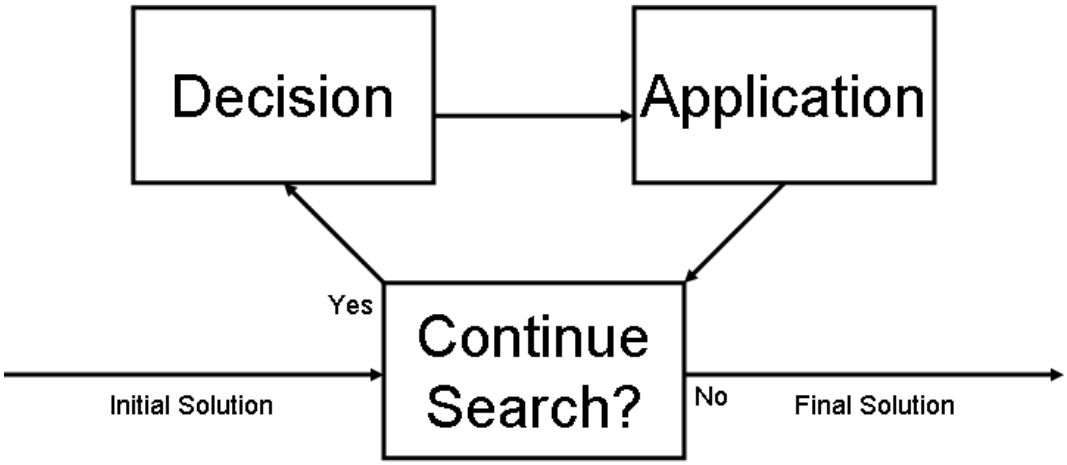

FIGURE 3.2: The Hyperheuristic Decision Cycle

the interaction each given heuristic will have with a given solution from that same domain. The hyperheuristic does not have access to this domain-specific information.

The amount of domain-free information available is limited. Knowing the evaluation values of the solutions encountered, the hyperheuristic can determine which of them is most optimal[2], whether a given heuristic has led the hyperheuristic to a better solution or not, and by how much the solution quality has changed.

The hyperheuristic may know whether any given solution is complete or partial,

[2]or which might constitute a member of a Pareto front, in the event of multiple criteria, which we do not consider in this thesis

feasible or infeasible, and possibly optimal or sub-optimal. The problems we have described earlier all involve complete solutions. It has been remarked that such factors should be taken into account by the hyperheuristic. The Boundary Problem, first remarked upon by Gaw et al. (authors of [GRK04]), points out that a compound evaluation function (such as the that of the Travelling Tournament Problem) applies a large weight to the hard constraint violations component, but the hyperheuristic has no way of knowing this. The danger is that a heuristic may cross the boundary (e.g. by finding a feasible neighbour of an infeasible solution) and be rewarded by a quantity corresponding to the quality improvement. The following search iterations may then be confused: the rewarded heuristic may be expected to produce an improvement of the same quantity which it can no longer achieve (and if the heuristic is designed to only find more feasible solutions, the heuristic may never be capable of producing an improvement), and equally the scale of the reward may prevent the hyperheuristic from selecting any other heuristics for a time. The magnitude of the reward may determine how long it will take for the hyperheuristic to adapt to its new environment. Such adaptation may be more costly if the search requires the hyperheuristic to reconsider solutions with hard constraint violations as means to escape local optima.

Not necessarily finally, the hyperheuristic may know how much processing time was spent applying the heuristic to the solution, and therefore which heuristics might be deemed quicker than others; the use of such heuristics might be promoted on the basis that more heuristic calls can be made in an equivalent time period, and conceivably more progress through a search space can be made. However, this has several drawbacks, not least of which that a heuristic which requires more processing time might find better specific

moves in the search space, justifying the time required for its use.

Furthermore, the amount of time taken does not depend on the nature of the heuristic, but on its implementation by a programmer and the programming language, compiler, operating system and hardware the program is to be run with. The heuristics used in the problems presented in this thesis are such that the CPU time required to implement most of them is recorded at less than 1 millisecond, which is the smallest unit of time this programmer could find to measure such things by (using the C++ *clock()* function).

The importance of each of these factors is difficult to determine. Certainly the knowledge of which solutions are most optimal is paramount to the overall running of the program, but as the solution space remains unknowable to the hyperheuristic, it cannot be known which (individual or combination) of the available heuristics, combined with which of the known solutions, is best suited to encounter the next most optimal solution.

The choice of which solution to further explore at any point is also difficult. Theoretically the hyperheuristic may memorise all the solutions it encounters and apply a valid heuristic upon any of them, but memory may be limited[3]. Many of the hyperheuristics described in Chapter 2 limit themselves to just a 'current' solution and one of its neighbours, which it may accept or reject. If the neighbour is accepted it becomes the 'current' solution of the following iteration.

The hypothesis of the choice function posits that different areas of a solution space might possess characteristics making some heuristics more useful than others, and that this usefulness would be measured in improvement. It can be readily imagined that heuristics

[3] A discussion of several facets of memory management in hyperheuristics, mostly concerning the relative importance placed on decisions made and solutions found recently as opposed to long ago in a search, can be found in [BBG+07]; the authors express concern that tuning an optimal importance parameter is a time-consuming task, sensitive to the problem instances under examination.

which intend to make infeasible solutions more feasible (by removing constraint violations, for example) will be more useful in areas of the solution space where solutions are infeasible, and these heuristics will be less useful in feasible areas.

3.5 On the Nature of Experiments

The primary objective of optimisation software is to find a solution of highest possible quality or lowest possible penalty, with the secondary objective that it must be found in a reasonable period of time. This remains true of software produced by hyperheuristic research. It is, however, very difficult to compare the performance of different algorithms, particularly where those algorithms have been written by different programmers with different levels of skill and different specific intentions, or where they are presented in papers with page limits and compromised priorities and perhaps there is only enough space for an objective value or CPU time for the best or average solution over a given number of experiments.

These data are misleading. Papers often do not specify the initial solutions (except where the technique is constructive, and the initial solution is implicitly empty, or the construction technique is deterministic), so it is difficult to begin within the same 'vicinity' as the discovered best solution. Equally, the duration of an experiment is dependent to some extent upon the speed of the computer running the program and the efficiency of the compilation. For example, it may be more efficient to evaluate the effect of a heuristic upon a solution and modify an objective value accordingly than to reevaluate the entire solution. The process of outputting data accumulated by a program to text files also significantly

extends the runtime of the program.

The platform on which the experiments are run is also a factor. The experiments described in this thesis are executed on Pentium IV 1.8GHz 256MB RAM machines running Windows XP Professional 2002 with Service Pack 2, using code written on and compiled with Visual Studio C++ .NET (version 7.1.3088). This platform is incapable of calculating a measurement of duration to a more exact precision than milliseconds; some of our heuristics appear to take less than one millisecond to run, but their durations appear as 0. Certain hyperheuristics must be adapted to avoid mathematical problems; the choice function hyperheuristic, for example, evaluates heuristics by dividing their improvement by their duration, so dividing by a duration of 0 is clearly impossible; we therefore assume that where results of applications of the choice function have been presented [Sou03], the program's heuristics were slower than ours. Where necessary a duration is assumed to be 1 (millisecond) to avoid mathematical upsets.

This leads us to a crucial decision for our experiments: what should our stopping criteria be? The slogan of hyperheuristics is "good enough, fast enough, cheap enough", directly implying that good results within a set reasonable amount of time are desired, and that a time limit is an appropriate stopping condition; this is corroborated by [CKS01a, KSC02], where a time limit was applied. But as already noted, a time limit is not of greatest use when we wish to reproduce the results of others, or present work which can be reproduced. The significant alternative is an evaluation limit, i.e. the hyperheuristic must produce its result within a given number of heuristic calls, or must terminate when a set number of heuristic calls fail to produce any better solutions; this provides reasonable

replicability at the cost of not knowing exactly when the experiment will end. This means that every heuristic call becomes more important; the experiments favour short sequences of heuristics which may be take a long time to apply over long sequences of heuristics which may be applied in a short period of time.

This makes it difficult to compare approaches with the literature. For example, we notice that in [Sou03], variations in parameters meant that experiments involving the Sales Summit Scheduling problem varied between 2300 and 9000 heuristic calls (Soubeiga does not specify) in the 600 second time limit. Soubeiga's results with the Project Presentation Scheduling problem also had a 600 second stopping condition, as examined in [CKS02], which was compared with Han's HyperCLGA's results following a preset number of generations [HK03b]; though a CPU time for the latter algorithm was provided, the variable length of the chromosomes in that algorithm mean that the total number of heuristic calls is impossible to determine. We use heuristic calls so that future researchers may replicate the results presented here.

3.6 Conclusion

We have presented two scheduling problems from the literature, as they will be acknowledged by the hyperheuristic framework. Results from previous approaches are presented as a measure of the competitiveness of our hyperheuristics. In the next chapter we present the algorithm of our new hyperheuristic.

Chapter 4

The Ant Algorithm Hyperheuristic: Transliteration

4.1 Introduction

Chapter 2 describes several hyperheuristics inspired by popular metaheuristic methods. In this chapter we introduce the first new hyperheuristic of this thesis, one based upon another popular metaheuristic method: the ant algorithm technique introduced by Dorigo et al. [DMC96].

As presented in that paper, the ant algorithm metaheuristic is inspired by the real-life means by which ants adaptively derive the shortest route between important locations such as their nest and their current food sources, and communicate this information between each other. The essential element of this communication is a substance known as *pheromone* which the ants lay as they travel, and therefore the presence of greater quantities of pheromone indicates a path well travelled; the pheromone decays over time, so a colony must continue to traverse the route in order to maintain it; routes which become poor (e.g.

if a food source runs out) can be left to decay and other routes will become more prominent.

The paper applied the technique to the Travelling Salesman Problem, a well known problem in the literature. Presented with a number of cities n and the $n \times n$ matrix d which provides the travelling cost between every pair of cities, the objective is to find the shortest route which visits every city once, returning to the starting city. Since even a symmetrical n-city TSP instance[1] has $\frac{n-1!}{2}$ possible routes[2], the problem is NP-hard.

The ant algorithm metaheuristic created a network in which vertices represented cities and the arcs between vertices represented the routes between cities. A colony of ants is then distributed among the vertices of the network, and they traverse the arcs of the network creating routes which can be evaluated at the end of the cycle when all of the tours are complete, i.e. comprise every city. Each ant uses the distance matrix d to probabilistically favour nearby cities over far away cities (as a heuristic evaluation of the problem domain, this is known as "visibility"), and they use pheromone laid down by the colony in previous cycles to favour arcs which were components of good tours over arcs which were not. The ants are aided in their navigation by the ability to recall which cities they have already visited and avoid them, restricting their choices and ensuring every cycle ends with feasible if poor solutions. As the cycles continue, it is hoped that the colony's "experience" directs the ants towards good long-term component arcs and away from poor arcs which appear good according to the domain information.

Our interest in the ant algorithm metaheuristic is its apparent qualities for cre-

[1] i.e., an instance with n cities and in which between every pair of cities i, j the travelling cost is the same in both directions, $d_{ij} = d_{ji}$

[2] Any city can be first since the tour is circular. There are $n-1$ potential second cities followed by $n-2$ potential third cities, etc., but since the problem is symmetrical $d_{ij} = d_{ji}$ and the tour can be navigated in either direction, we divide by $2n$

ating good sequences of moves. As a hyperheuristic is a constructor of algorithms, we are interested in exploring and building good sequences of heuristics to apply. To begin our investigation, we transliterate the ant algorithm metaheuristic as closely as we can to create our first ant algorithm hyperheuristic.

4.2 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, with any excess ants being distributed randomly.

The ants then construct a sequence of heuristics by traversing the network. At each decision point (or *step*), 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 sequence (i.e. in graph terminology, the sequence is a *walk*).

After each ant has visited a certain number of heuristics, the ant pauses to analyse the walk it has just traversed and to lay an amount of pheromone on each arc 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 use the term *cycle* to specify the time taken between all ants beginning their paths and all ants completing their paths, and the

algorithm continues for as many cycles as is required.

The choice of heuristic at each step is determined by the combination of visibility information and pheromonal experience, which, if we follow Dorigo's example, are defined by the domain. This is problematic, since our hyperheuristic framework has no domain knowledge. The domain is of low-level heuristics and their ability to improve a solution, which cannot be known in advance and must be predicted. We therefore choose to apply the terms visibility and pheromone to their short- and long- term potential, respectively: the visibility of heuristic j, which we write as η_j, represents the uniformly distributed current confidence that heuristic j will lead to a good solution, and the pheromone on the arc between heuristics i and j, which we write as τ_{ij}, represents the confidence that applying heuristics i and j in that order will lead to a good solution (illustrated in 4.1).

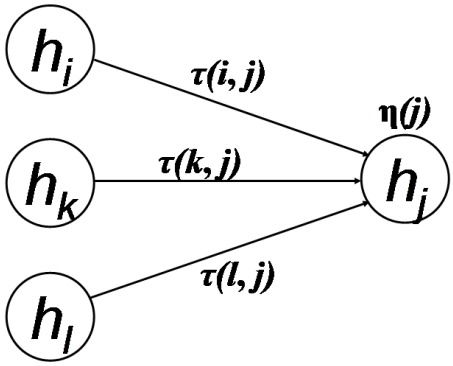

FIGURE 4.1: The information available at a decision point to guide the search to heuristic h_j.

Each cycle becomes an exploration of many heuristic routes through the solution space, one route per ant. By the end of the cycle, a set of new solutions will have been discovered, and a new best solution may have been found. Unlike the metaheuristic, the

ants cannot simply reset to a blank solution and a random vertex from which to begin a new route at the beginning of the new cycle; since each sequence begins with a known solution, we must continue to evaluate further moves from that solution and the vertex representing the heuristic which discovered it, which becomes the starting point for new routes from that solution.

Initial experiments assumed that all ants kept their solutions from cycle to cycle, but taking their direction from the colony's accumulated visibility and pheromone experiences. These experiments performed rather poorly, and it was supposed that a colony widespread across a large area of the solution space might in fact be providing contradictory information on the heuristics they encountered because of their differing experiences. Better results were found when the ants regrouped at the end of each cycle to the best solution found by the colony in that cycle, and it was hypothesised that by restricting the colony to a given area their collective information was being kept relevant and productive to all. In practice this means that at the beginning of a cycle, the ants all relocate in the solution space to the best solution found during the previous cycle and in the heuristic space to the heuristic which discovered that solution.

We also consider that at the decision level an ant may find it useful to choose to reject a new solution it discovers if the new solution is poorer than the ant's current solution, and backtrack: the walk taken by the ant through the heuristic space is less important than the exploration of the solution space, and if a walk is poor then this ability may be required to direct the ant to better places. For terminology purposes we refer to the ant's movements through the heuristic space as a *journey* which may be one walk, as intended, or may be

several branched walks if decisions have been backtracked. A cycle is measured by the size of the journeys, not walk-lengths: the solutions at the end of the cycle may be one - or not even one - heuristic move away from the solution at the beginning of the cycle if the solutions discovered are poor.

For the purposes of learning, 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 journey arc (x, z) and not (x, y) or (y, z).

We use improvement as our acceptance criterion: if the new solution is an improvement, the solution is accepted; if the new solution is not an improvement, the solution is rejected. Without this criterion each ant / hyperheuristic agent 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 / hyperheuristic agent is an Only Improving (OI) Hyperheuristic. We distinguish here between hyperheuristic agents and the overall hyperheuristic since the new cycle always begins from the best solution of the previous cycle, allowing an Any Moves Ant Algorithm Hyperheuristic some form of quality control. Previous research [Sou03] 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 difference in performance is sensitive to the problem

domain and the available heuristics.

4.3 Pseudocode

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

1. **Initialise**
   Initialise all variables.
   Initialise a solution S_0, and set best solution $S_b := S_i$.
   Set $t := 0$. {t is the heuristic calls counter}.
   Set $c := 0$. {c is the cycle counter}.
   Set n to be the number of available low level heuristics. Create a connected network with n corresponding vertices.
   Set m to be the number of ants. Scatter the ants uniformly on the n vertices.
   For every vertex $j := 1$ to n do
   - Set the initial value of $\eta_j = 0$. {Initialise visibility.}

   Set LJ to be the length of an ant's journey, and set $p := 0$. {Each cycle after the first cycle can be measured as $LJ \cdot m$ heuristic calls in duration. p counts the number of heuristic calls that each ant has taken.}

2. **First Cycle**
   We seek to create initial visibility values, and find a solution S_1 to begin the search. The ants have no involvement in this cycle.
   For $j := 1$ to n do
   - Apply heuristic j to solution S_0. Let the resulting solution be S_j.
   - Update the visibility of heuristic j by updating η_j according to equation 4.1.
   - If $j = 1$ or if S_j is better than S_1, let $S_1 := S_j$, such that S_1 is the best solution one heuristic move from S_0. Let the starting vertex of the new cycle $h_1 := j$.
   - If S_1 is better than S_b let $S_b := S_1$.

   For $k := 1$ to m do
   - Provide the kth ant with a solution S_k such that $S_k := S_1$.
   - Move the kth ant to vertex $h_k := h_1$.

   Set $t := t + m$.
   Set $c := 1$.
   Create a null solution S_c and null heuristic value h_c to store the heuristic which discovered solution S_c.
   For every pair of vertices i, j, set an initial pheromone value to edge (i, j), $\tau_{ij}(t) := 0$.

3. **Heuristic Exploration**
   In this phase, heuristics are chosen and applied, and solutions are accepted for further exploration or rejected. All decisions use the same pheromone and visibility information.
   For $k := 1$ to m do

- If ($t = t_{max}$)
  - Go to step 6.
- At decision point t, the kth ant is on vertex h_k. Set vertex $i := h_k$.
- Choose the heuristic j for ant k to move to, with probability $probability_{ijk}(t)$ given by equation 4.7.
- Apply heuristic j to solution S_k to produce S'_k.
  The new solution is then evaluated for further exploration by this ant and by the colony, and also saved if it is the best solution found so far.
- If (S_c is null or S'_k is better than S_c)
  - Set $S_c := S'_k$. Set $h_c := j$
- If (Only Improvement & S'_k is worse than S_k)
  then
  - Reject S'_k.

  else
  - Accept S'_k.
  - Move ant k to vertex j {i.e. set $h_k := j$}.
  - Set $S_k := S'_k$.
- If (S'_k is better than S_b)
  - Set $S_b := S'_k$.

4. **Visibility Update**
   - For $k := 1$ to m do
     - Update η_j according to equation 4.1.
   - Set $p := p + 1$. {The ants have moved one step along their path.}
   - Set $t := t + m$.
5. **Pheromone Update**
   - If ($LJ = p$) {The ants have reached the end of their path.}
     - Update $\tau_{ij}(t)$ for all accepted arcs (i, j) according to equation 4.2.
     - Set $S_c := S_b$ {Select the best solution found during this cycle, for further exploration in the next cycle}.
     - For all ants, set $h_k := h_b$. {All ants begin the next cycle at the vertex which produced their new current solution}.
     - Set $c := c + 1$.
     - Set $p := 0$. {Reset for new cycle.}
     - Go to step 3.
6. **Stopping Condition**
7. Output best solution S_b.
8. Stop.

The approach is illustrated in Figures 4.2, 4.3, 4.4, 4.5 and 4.6. The colony arrives in an area of the solution space, explores outwards in many different directions, communicate their findings, and swarm to the most interesting (best quality) area found. If such an area is not found the colony remains where they are and continues to explore. The higher the number of ants, or the longer the colony remains in one area, the more intense the search in that area, and equally the less the hyperheuristic sees of the overall search space.

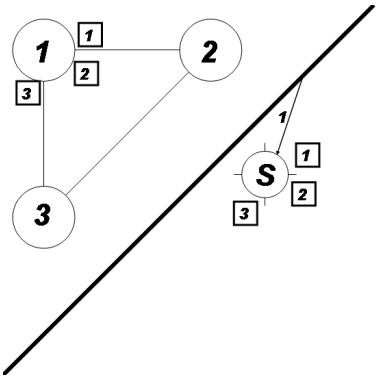

FIGURE 4.2: Beginning of cycle ($t = 0$): Three ants (small squares, numbered) located at current best solution S in the solution space (bottom right of diagonal line in figures) and the heuristic h_1 (top-left numbered vertex of graph) in the heuristic space (top left of diagonal line in figures) which discovered solution S.

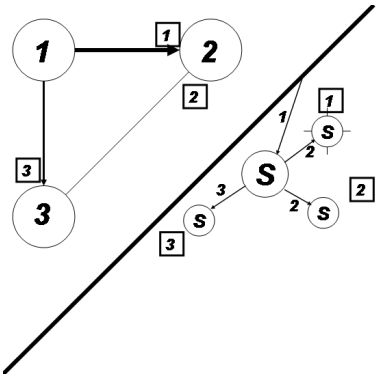

FIGURE 4.3: The ants explore ($t = m$): Ants a_1 and a_2 choose heuristic h_2 and ant a_3 chooses heuristic h_3. Ant a_1 discovers a new best solution (indicated by crosshairs). Visibility is updated for heuristics h_2 and h_3.

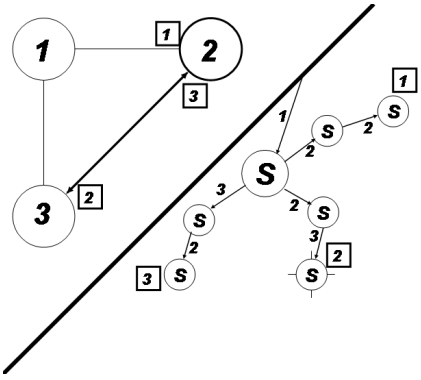

FIGURE 4.4: The ants explore ($t = 2m$): Ant a_1 stays at heuristic h_2, a_2 moves to h_3 and a_3 moves to h_2. a_2 discovers a new best solution (indicated by relocated crosshairs). Visibility is updated for heuristics h_2 and h_3.

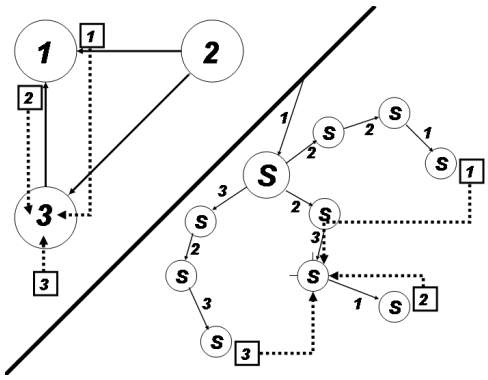

FIGURE 4.5: End of cycle ($t = 3m = ml$): Ants a_1 and a_2 returns to h_1 while ant a_3 returns to h_3. Visibility is updated for heuristics h_1 and h_3. No new best solution is found. Pheromone is laid on paths a_1 (1-2-2-1), a_2 (1-2-3-1) and a_3 (1-3-2-3). The ants relocate to new best solution S discovered in the solution space and the heuristic which discovered S, h_3, in the heuristic space.

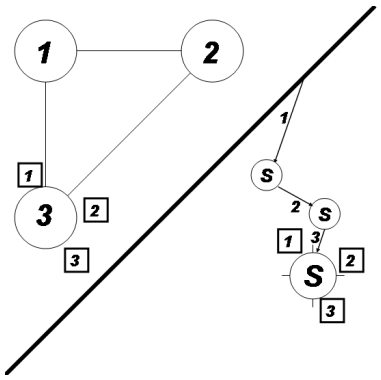

FIGURE 4.6: Beginning of next cycle ($t = ml$): Three ants at current best solution S in the solution space and the heuristic which discovered S in the heuristic space, h_3.

We use a visibility function inspired by the choice function hyperheuristic [CKS01a, CKS02, Sou03], 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:

$$\eta_j(t) = \gamma\eta_j(t-m) + \sum_{k}^{m} \frac{I_{kj}(t)}{T_{kj}(t)} \tag{4.1}$$

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 amount of CPU time 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. [Sou03]) 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 and ρ for the corresponding aspects 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 [Sou03] 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 anticipate 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 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:

$$\tau_{ij}(t) = (1 - \rho)\tau_{ij}(t - m \cdot n) + \sum_{k}^{m} \frac{\#_{ij}(P_k(t)) \cdot I(P_k(t))}{T(P_k(t))} \tag{4.2}$$

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 time. 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"

[CKS01a]. We first calculate a value V for each heuristic j, from the previous heuristic i, using the formula:

$$V_{ij}(t) = \alpha\eta_j(t) + \beta\tau_{ij}(t) \tag{4.3}$$

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

$$PV_{ij}(t) = max\{V_{ij}(t), Q\sigma^{V_{ij}(t)}\} \tag{4.4}$$

where

$$Q = \frac{\sum_h max\{0, V_{ih}(t) + \epsilon\}}{10 \cdot n} \tag{4.5}$$

(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 ∞.

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.

$$PV_{ij}(t) = max\{max_{h \in H} PV_{ih}(t), 0\} \tag{4.6}$$

If none of the heuristics are performing well ($Q = 0$), 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

$$probability_{ijk}(t) = \frac{PV_{ij}(t)}{\sum_{h \in H} PV_{ih}(t)} \tag{4.7}$$

4.4 Experiments and Results

4.4.1 Stopping Condition

The stopping condition used for the Project Presentation Scheduling Problem in [CKS02, Sou03] was 600 seconds of CPU time on PC Pentium III 1.0GHz with 128MB RAM. In [HK03b] the stopping condition was 200 generations of a genetic algorithm hyperheuristic in which chromosome length was variable (the average CPU time came to 924 seconds). Since in neither case was the number of heuristic calls made clear, we set a stopping condition which can be easily duplicated by interested future researchers: 2000 heuristic calls, which corresponded to 25-100 seconds using our current implementation. The disparity can be accounted for by the output files our code produces in proportion to the number of heuristic calls the hyperheuristics accept; without such output the experiments last approximately 13-24 seconds. The number of heuristic calls was determined by experimentation; it was observed at 1000 calls that this number was insufficient to assess the learning capabilities of the hyperheuristics.

There is no set stopping condition in the literature for the Travelling Tournament Problem; most comparisons between approaches consider the best results found for each problem instance. [Che07] provides a series of stopping conditions measured in seconds of CPU time, but no arguments are provided as to why these times are considered “reasonable”; however, in at least one instance (*NL6*) the optimal solution is found within 250 cycles,

which in that approach corresponds to 25,000 heuristic calls. Since our approach does not utilise the greedy heuristics used by [Che07], we set a uniform stopping condition of 50,000 heuristic calls for all problem instances. This corresponds to a CPU time of 10-120 seconds using our current implementation.

The question of what constitutes a 'reasonable time' has no easy answer. We set an arbitrarily uniform number of heuristic calls for all instances rather than an a series of arbitrary times to more easily compare the hyperheuristics' performance with different problem sizes.

4.4.2 Parameters

The networks traversed by the ants correspond in size to the number of heuristics used by the hyperheuristic. As a default we use a number of ants equal to the number of vertices in the theory that this will allow each heuristic to be explored equally at first, and a journey length equal to the number of vertices in the theory that each ant may visit each heuristic equally during a cycle. We then vary these two parameters.

In the ant algorithm metaheuristic the journey length is often specifically related to the problem being solved, so we cannot use the metaheuristic as a predictor of how the hyperheuristic's performance will vary as its journey length varies. The number of ants, however, has been explored in the literature to determine how well they converge in the metaheuristic method [DMC96, BF03]. It is commonly understood that an optimal number of ants will exist to converge upon good solutions in fewer cycles: if the colony is smaller than this number, convergence will take longer (or may never occur); if the colony is larger, convergence takes longer simply because of the extra processing requirements of the extra

ants.

We might predict, however, that a larger number of ants will find better solutions than a smaller number of ants. Certainly we can anticipate that a larger number of ants will intensify the search around a particular area of the solution space, and a larger journey length will diversify the search area, considering many options in some definition of depth, meaning the colony can cross the solution space with some higher measure of confidence in its progress. However, both variables are constrained by the stopping condition of the search, and the nature of the solution space: a search with fewer ants or shorter journey lengths will be more directed, perhaps taking better advantage of areas of the search space where improvement is easy and there is less need to intensify or diversify. The unpredictability of the solution space also means that the heuristic network may never converge.

We use as variables the values 1, 2, 3, 4, 5, 6, 8, 10, 12 and 15. For each experiment this value will correspond to either the journey length or the number of ants, with the other variable assuming the number of heuristics as its value (8 for the PPSP, 5 for the TTP).

Every hyperheuristic was run 25 times on each of the problem instances described for the Project Presentation Scheduling Problem and Travelling Tournament Problem in chapter 3.

The hyperheuristics are the Simple Random Hyperheuristic (SR) presented in [CKS02, Sou03] for comparative purposes and our Ant Algorithm Hyperheuristic with journey lengths of 1-15 heuristic calls, or 1-15 ants, using the above parameter range. The hyperheuristics were also distinguished by their solution acceptance criteria: either the hyperheuristic accepts Any Move (AM), or it accepts only improving moves (OI)[3].

[3]Two probabilistic acceptance criterions were considered. The first was a simulated annealing criterion,

We assign to α, β, γ, ρ respectively the values 1, 1, $\frac{1}{n^b}$, $\frac{1}{n^b}$. Early experiments did not show any discernable patterns between the weights of visibility and pheromone, so we assign both equal weights. We use a recency weight equal to $\frac{1}{n}$ raised to the power of b, where b is the number of heuristics not used during that particular step or cycle respectively: this is intended to curb any excessive convergence towards particular heuristics or arcs.

The results presented in Tables 4.1-4.9 show the worst, mean average and best results, with standard deviation, from each algorithm applied to the three problem instances. The initial solution quality for each instance is provided at the top of the respective problem instance's column.

4.5 Analysis

It is evident that our results are an improvement upon previous results with the Project Presentation Scheduling Problem, and they are competitive with prior results with the Travelling Tournament Problem. It is also evident that the Ant Algorithm Hyperheuristic results are not significantly better than the Simple Random Only Improving Hyperheuristic.

The most significant parameter seems to be the acceptance criterion. For smaller instances of the Travelling Tournament Problem (*NL6*, *NL8*), better results were produced by the Ant Algorithm Hyperheuristic variants which accepted Any Move. We hypothesise that in smaller instances, where each heuristic move has a more significant effect upon the whole solution, that navigating between poor and good solutions may be a less gradual and more complex process (since it is more difficult to rearrange combinations of matches) and

using the same temperature function used by Bai and Kendall [BK03], but the results were not promising, perhaps because the evaluation functions for our problems are not similar to those investigated by Bai and Kendall. The second considered a uniform 0.1 probability of accepting a poorer solution, but this also did not perform well. We therefore present the 'extreme' criteria in more detail.

TABLE 4.1: Results of hyperheuristic experiments on PPSP csit0

PPSP csit0		Initial Solution Quality			-923.5
Algorithm		Worst	Average	Best	Std Dev
Simple Random					
	AM	-1435.2	-1501.91	-1572.3	40.8726
	OI	-1577.5	-1628.31	-1679.5	25.8409
Ant Algorithm					
Walk 1	AM	-1591.7	**-1651.09**	-1705.5	28.8323
Walk 2	AM	-1571.5	-1633.3	-1706.5	28.6762
Walk 3	AM	-1556.6	-1625.38	-1685.5	27.3817
Walk 4	AM	-1540.5	-1604.09	-1656.6	22.8344
Walk 5	AM	-1526.5	-1599.48	-1647.	36.5515
Walk 6	AM	-1544.5	-1605.94	-1654.7	27.1519
Walk 8	AM	-1510.5	-1585.21	-1634.6	28.3266
Walk 10	AM	-1535.7	-1592.14	-1634.8	24.1182
Walk 12	AM	-1552.6	-1591.65	-1621.9	19.5958
Walk 15	AM	-1517.5	-1573.03	-1601.7	21.0011
Walk 1	OI	-1568.5	-1631.02	-1706.5	32.0964
Walk 2	OI	-1594.5	-1634.64	-1676.5	23.1991
Walk 3	OI	**-1596.5**	-1630.14	-1674.5	20.2975
Walk 4	OI	-1578.5	-1621.6	-1684.	27.0170
Walk 5	OI	-1583.5	-1624.6	-1670.5	20.3726
Walk 6	OI	-1587.5	-1622.62	-1682.5	27.3242
Walk 8	OI	-1541.5	-1608.3	-1644.5	25.9487
Walk 10	OI	-1550.5	-1607.71	-1651.5	23.6604
Walk 12	OI	-1544.5	-1599.02	-1654.5	27.5774
Walk 15	OI	-1526.5	-1595.67	-1628.6	25.5680
Ants 1	AM	-1444.8	-1527.07	-1626.6	47.1358
Ants 2	AM	-1501.4	-1562.39	-1620.	29.8085
Ants 3	AM	-1556.6	-1587.5	-1639.9	23.2994
Ants 4	AM	-1562.6	-1591.33	-1624.9	18.7289
Ants 5	AM	-1567.3	-1595.91	-1636.8	19.4275
Ants 6	AM	-1564.5	-1598.46	-1646.5	20.4436
Ants 8	AM	-1510.5	-1585.21	-1634.6	28.3266
Ants 10	AM	-1542.5	-1589.34	-1628.7	20.8125
Ants 12	AM	-1543.5	-1589.98	-1626.7	19.7295
Ants 15	AM	-1507.6	-1569.45	-1608.7	21.7556
Ants 1	OI	-1571.5	-1638.62	-1694.5	29.0174
Ants 2	OI	-1581.5	-1637.27	-1689.5	25.7529
Ants 3	OI	-1588.5	-1627.56	**-1711**.	29.2148
Ants 4	OI	-1551.5	-1623.06	-1686.5	31.7321
Ants 5	OI	-1563.5	-1618.66	-1707.6	32.3414
Ants 6	OI	-1557.5	-1615.42	-1669.5	23.4018
Ants 8	OI	-1541.5	-1608.3	-1644.5	25.9487
Ants 10	OI	-1562.5	-1601.42	-1642.5	20.4047
Ants 12	OI	-1553.5	-1591.34	-1633.5	20.0347
Ants 15	OI	-1529.6	-1582.14	-1622.5	20.6793

TABLE 4.2: Results of hyperheuristic experiments on PPSP csit1

PPSP csit1		Initial Solution Quality			-2569.2
Algorithm		Worst	Average	Best	Std Dev
Simple Random					
	AM	-2565.2	-2640.36	-2733.4	41.4692
	OI	-2979.5	-3014.97	-3057.2	20.8223
Ant Algorithm					
Walk 1	AM	-2948.9	-2994.38	-3042.4	24.9487
Walk 2	AM	-2940.4	-2996.28	-3045.8	28.4924
Walk 3	AM	-2898.3	-2977.08	-3018.5	28.474
Walk 4	AM	-2910.8	-2970.38	-3009.9	28.9758
Walk 5	AM	-2917.9	-2971.33	-3007.2	19.5272
Walk 6	AM	-2898.8	-2956.28	-2999.5	25.3555
Walk 8	AM	-2903.8	-2958.22	-3011.2	26.4597
Walk 10	AM	-2897.5	-2946.39	-3019.7	27.4118
Walk 12	AM	-2886.2	-2938.35	-2973.8	22.0361
Walk 15	AM	-2891.8	-2932.19	-2963.7	20.2987
Walk 1	OI	-2985.1	**-3018.64**	**-3055.1**	19.8189
Walk 2	OI	-2965.8	-3014.63	-3051.4	23.0934
Walk 3	OI	-2971.4	-3016.72	-3048.1	20.4465
Walk 4	OI	**-2988.7**	-3017.58	-3043.8	15.4106
Walk 5	OI	-2946.5	-3002.6	-3035.8	22.7509
Walk 6	OI	-2966.4	-3011.83	-3038.6	17.7385
Walk 8	OI	-2958.1	-3007.08	-3046.5	23.6488
Walk 10	OI	-2951.5	-3004.02	-3033.5	18.5417
Walk 12	OI	-2926.6	-2999.24	-3046.9	24.7890
Walk 15	OI	-2925.6	-2988.78	-3049.9	26.6906
Ants 1	AM	-2696.	-2788.36	-2912.6	55.1199
Ants 2	AM	-2818.6	-2884.95	-2930.4	30.9963
Ants 3	AM	-2875.7	-2922.73	-2986.1	30.2043
Ants 4	AM	-2899.	-2929.41	-2976.6	21.8730
Ants 5	AM	-2894.5	-2946.56	-2993.9	21.8722
Ants 6	AM	-2893.9	-2942.77	-2990.5	19.6484
Ants 8	AM	-2903.8	-2958.22	-3011.2	26.4597
Ants 10	AM	-2892.5	-2948.15	-2982.0	24.2996
Ants 12	AM	-2884.1	-2951.58	-2991.1	22.9304
Ants 15	AM	-2902.1	-2952.02	-3002.9	24.6677
Ants 1	OI	-2972.	-3012.42	-3053.1	21.0041
Ants 2	OI	-2972.8	-3009.88	-3043.3	19.3318
Ants 3	OI	-2966.5	-3011.8	-3045.9	21.4388
Ants 4	OI	-2963.8	-3014.28	-3043.5	20.6366
Ants 5	OI	-2962.8	-3014.47	-3045.2	20.7113
Ants 6	OI	-2940.3	-3009.	-3046.9	25.193
Ants 8	OI	-2958.1	-3007.08	-3046.5	23.6488
Ants 10	OI	-2952.4	-3004.11	-3028.9	19.6163
Ants 12	OI	-2969.7	-3000.06	-3048.8	20.7332
Ants 15	OI	-2952.4	-2997.97	-3040.2	20.1767

TABLE 4.3: Results of hyperheuristic experiments on PPSP csit2

PPSP csit2		Initial Solution Quality			-950.1
Algorithm		Worst	Average	Best	Std Dev
Simple Random					
	AM	-947.1	-1227.42	-1459.1	110.7607
	OI	-1769.0	-1805.61	-1849.7	23.4332
Ant Algorithm					
Walk 1	AM	-1772.1	-1812.11	-1852.	22.6337
Walk 2	AM	-1758.8	-1802.94	-1868.7	30.4228
Walk 3	AM	-1742.	-1799.47	-1854.5	24.2981
Walk 4	AM	-1709.5	-1787.78	-1862.	36.1648
Walk 5	AM	-1726.6	-1784.28	-1846.1	33.8598
Walk 6	AM	-1689.2	-1777.11	-1837.8	37.8003
Walk 8	AM	-1732.5	-1783.29	-1824.7	23.0408
Walk 10	AM	-1655.1	-1768.45	-1821.9	39.1636
Walk 12	AM	-1658.7	-1756.64	-1828.4	38.3514
Walk 15	AM	-1696.7	-1753.28	-1780.2	21.7413
Walk 1	OI	-1774.4	-1822.76	-1853.5	18.4260
Walk 2	OI	-1798.3	**-1832.47**	-1863.1	18.3761
Walk 3	OI	**-1798.4**	-1830.57	-1869.2	17.7299
Walk 4	OI	-1776.1	-1828.49	**-1884.7**	24.4283
Walk 5	OI	-1754.2	-1818.01	-1870.8	27.1554
Walk 6	OI	-1755.4	-1818.52	-1860.4	26.2754
Walk 8	OI	-1726.4	-1805.9	-1871.9	35.1381
Walk 10	OI	-1715.	-1807.11	-1872.7	38.1785
Walk 12	OI	-1631.1	-1805.08	-1860.9	49.2653
Walk 15	OI	-1732.2	-1807.68	-1860.5	33.7839
Ants 1	AM	-1425.6	-1562.18	-1690.1	67.4259
Ants 2	AM	-1624.5	-1692.21	-1789.1	44.1620
Ants 3	AM	-1669.3	-1730.62	-1786.	28.0092
Ants 4	AM	-1634.8	-1743.1	-1801.8	39.5615
Ants 5	AM	-1698.5	-1761.58	-1817.1	29.4995
Ants 6	AM	-1748.7	-1772.91	-1808.1	17.2012
Ants 8	AM	-1732.5	-1783.29	-1824.7	23.0408
Ants 10	AM	-1718.4	-1779.78	-1848.7	34.6732
Ants 12	AM	-1708.8	-1786.	-1841.7	33.5877
Ants 15	AM	-1728.	-1780.16	-1827.1	27.1093
Ants 1	OI	-1732.	-1798.1	-1836.4	24.8418
Ants 2	OI	-1781.	-1815.52	-1849.7	18.7222
Ants 3	OI	-1773.5	-1821.26	-1873.4	24.0217
Ants 4	OI	-1768.4	-1819.25	-1854.7	20.8857
Ants 5	OI	-1744.9	-1819.62	-1883.3	28.4655
Ants 6	OI	-1727.3	-1813.94	-1879.7	32.8732
Ants 8	OI	-1726.4	-1805.9	-1871.9	35.1381
Ants 10	OI	-1706.	-1805.8	-1861.1	39.5718
Ants 12	OI	-1781.4	-1827.64	-1863.7	19.5716
Ants 15	OI	-1696.2	-1804.83	-1858.4	35.3243

TABLE 4.4: Results of hyperheuristic experiments on TTP NL6

TTP NL6		Initial Solution Quality				14543529
Algorithm		Feasible	Worst	Average	Best	Std Dev
Simple Random						
	AM	1	28079	27476.88	26624	437.2495
	OI	1	27751	25830.4	24764	795.2817
Ant Algorithm						
Walk 1	AM	1	26003	25320.	24863	255.2966
Walk 2	AM	1	25638	25138.96	24414	318.4340
Walk 3	AM	1	25752	25146.52	24073	398.1296
Walk 4	AM	1	25915	25245.76	24482	369.6499
Walk 5	AM	1	25876	25130.92	24174	346.7888
Walk 6	AM	1	25814	25317.16	24819	294.4586
Walk 8	AM	1	26056	25385.44	24341	392.8626
Walk 10	AM	1	25848	25290.72	24482	315.7063
Walk 12	AM	1	26365	25461.68	24467	367.7022
Walk 15	AM	1	26019	25454.76	24568	403.2211
Walk 1	OI	1	28312	26364.56	24912	876.5729
Walk 2	OI	1	27514	26203.48	25111	656.0087
Walk 3	OI	1	28772	26332.56	24702	1082.375
Walk 4	OI	1	27898	26187.16	24897	697.3544
Walk 5	OI	1	28032	26136.8	25024	790.8755
Walk 6	OI	1	28647	26313.88	24660	953.6941
Walk 8	OI	1	28756	26464.24	24872	1137.7692
Walk 10	OI	1	27374	25976.92	24085	1001.0698
Walk 12	OI	1	28148	26307.52	24691	847.4637
Walk 15	OI	1	28237	26174.32	24498	928.7251
Ants 1	AM	1	27564	26800.56	25673	521.7133
Ants 2	AM	1	26865	26085.52	25308	343.0368
Ants 3	AM	1	26216	25763.56	25217	282.3181
Ants 4	AM	1	26176	25477.16	24787	329.7352
Ants 5	AM	1	25876	25130.92	24174	346.7888
Ants 6	AM	1	25541	25033.8	24122	350.5503
Ants 8	AM	1	25241	24904.56	24101	286.3545
Ants 10	AM	1	25155	24768.92	24101	235.8917
Ants 12	AM	1	24994	24567.88	**23916**	376.2859
Ants 15	AM	1	**24895**	**24508.04**	24073	338.5663
Ants 1	OI	1	29738	26474.	24930	1104.9491
Ants 2	OI	1	28443	26462.68	25226	846.0113
Ants 3	OI	1	28170	26189.68	24757	905.6618
Ants 4	OI	1	27147	26010.48	24316	752.8346
Ants 5	OI	1	28032	26136.8	25024	790.8755
Ants 6	OI	1	28248	26154.	24122	969.1071
Ants 8	OI	1	28897	26235.44	24764	947.4623
Ants 10	OI	1	28212	26133.36	24961	918.9631
Ants 12	OI	1	28642	26464.76	24660	868.2422
Ants 15	OI	1	28170	26213.28	25008	760.5940

TABLE 4.5: Results of hyperheuristic experiments on TTP NL8

TTP NL8		Initial Solution Quality				37843485
Algorithm		Feasible	Worst	Average	Best	Std Dev
Simple Random						
	AM	0.88	64066	54563.31818	51458	1971.1537
	OI	1	49276	46762.4	44597	1468.8997
Ant Algorithm						
Walk 1	AM	1	49744	48496.04	46978	708.375
Walk 2	AM	1	49337	48482.2	47029	548.7991
Walk 3	AM	1	49820	48649.48	47415	593.0757
Walk 4	AM	1	49654	48549.48	46961	623.9252
Walk 5	AM	1	50004	48825.28	48109	393.2408
Walk 6	AM	1	49161	48450.64	47023	541.109
Walk 8	AM	1	50386	48786.72	46523	858.3282
Walk 10	AM	1	50197	49123.48	47861	671.3261
Walk 12	AM	1	50478	49361.92	48132	644.4752
Walk 15	AM	1	50386	49151.32	47857	625.111
Walk 1	OI	1	49539	46337.72	43367	1357.758
Walk 2	OI	1	49282	46484.56	44120	1412.441
Walk 3	OI	1	49063	46517.24	44105	1336.479
Walk 4	OI	1	49428	46870.36	43583	1360.237
Walk 5	OI	1	49632	46806.24	43851	1569.536
Walk 6	OI	1	50036	47154.	43495	1939.19
Walk 8	OI	1	49092	46166.	43918	1376.718
Walk 10	OI	1	48560	46466.6	44543	1110.464
Walk 12	OI	1	48074	46441.76	44340	1125.66
Walk 15	OI	1	48444	46342.64	43999	1090.929
Ants 1	AM	1	53277	51924.12	49866	1007.496
Ants 2	AM	1	52276	50451.6	49005	827.0229
Ants 3	AM	1	50594	49650.44	48232	730.5825
Ants 4	AM	1	50607	49242.16	48003	556.4624
Ants 5	AM	1	50004	48825.28	48109	393.2408
Ants 6	AM	1	49558	48511.32	47319	512.2413
Ants 8	AM	1	48841	47732.12	46392	652.8189
Ants 10	AM	1	48038	46999.76	45259	604.0355
Ants 12	AM	1	47899	46528.68	44352	792.7951
Ants 15	AM	1	**46543**	**45549.32**	43799	661.0142
Ants 1	OI	1	48554	46379.72	43418	1355.602
Ants 2	OI	1	49992	46987.96	45344	1053.496
Ants 3	OI	1	50543	46516.6	43617	1798.061
Ants 4	OI	1	50019	46565.92	42268	1663.134
Ants 5	OI	1	49632	46806.24	43851	1569.536
Ants 6	OI	1	48889	46503.56	43875	1473.801
Ants 8	OI	1	48629	46439.6	43475	1299.068
Ants 10	OI	1	48157	46452.56	43231	1272.165
Ants 12	OI	1	51140	46721.96	**41927**	1919.073
Ants 15	OI	1	49133	46699.88	44567	1244.145

TABLE 4.6: Results of hyperheuristic experiments on TTP NL10

TTP NL10		Initial Solution Quality				109205729
Algorithm		Feasible	Worst	Average	Best	Std Dev
Simple Random						
	AM	0	N/A	N/A	N/A	N/A
	OI	1	78223	74621.44	71751	1430.4891
Ant Algorithm						
Walk 1	AM	1	82888	81461.36	78501	1069.2986
Walk 2	AM	1	82148	80747.28	78791	903.4730
Walk 3	AM	1	82016	80919.12	79534	564.0752
Walk 4	AM	1	82790	80996.08	78849	1179.3563
Walk 5	AM	1	83761	81527.04	78013	1200.7146
Walk 6	AM	1	82985	81485.92	79049	890.6493
Walk 8	AM	1	83867	81623.64	78834	1277.2395
Walk 10	AM	1	83980	82038.88	78825	1331.2025
Walk 12	AM	1	83696	82102.76	79608	1146.6463
Walk 15	AM	1	84429	82842.84	80644	984.6959
Walk 1	OI	1	78122	74416.88	70398	1926.2356
Walk 2	OI	1	77711	74035.84	71844	1525.2572
Walk 3	OI	1	77894	74800.64	70862	1804.0835
Walk 4	OI	1	79248	74977.84	71822	2072.2391
Walk 5	OI	1	78451	73896.44	70326	2289.2381
Walk 6	OI	1	78034	74842.8	69371	2332.5750
Walk 8	OI	1	77335	74845.76	70590	1900.7698
Walk 10	OI	1	**76976**	74367.36	70944	1764.5179
Walk 12	OI	1	77355	73746.68	**69232**	2004.0665
Walk 15	OI	1	80661	75120.56	70350	2350.3205
Ants 1	AM	0.4	91732	88827.9	85445	2187.8354
Ants 2	AM	1	87958	85769.96	81536	1730.7096
Ants 3	AM	1	86225	83836.56	81190	1272.6047
Ants 4	AM	1	84075	82340.72	80651	994.3796
Ants 5	AM	1	83761	81527.04	78013	1200.7146
Ants 6	AM	1	82124	80117.12	77757	1017.1860
Ants 8	AM	1	80757	79291.64	77296	943.5420
Ants 10	AM	1	79573	78430.2	76825	833.2785
Ants 12	AM	1	78885	77522.96	75218	1038.3874
Ants 15	AM	1	79047	76225.68	72427	1486.4667
Ants 1	OI	1	78335	74523.52	69524	1903.2437
Ants 2	OI	1	77628	74273.12	70962	2142.0537
Ants 3	OI	1	77753	**73574.04**	70183	1891.2578
Ants 4	OI	1	79495	74811.08	71225	1865.0947
Ants 5	OI	1	78451	73896.44	70326	2289.2381
Ants 6	OI	1	79131	74401.48	70288	1906.7778
Ants 8	OI	1	78373	74185.04	70847	1800.6708
Ants 10	OI	1	78125	74369.6	71767	2034.4617
Ants 12	OI	1	78555	74686.32	69393	2216.9293
Ants 15	OI	1	79526	74175.84	71343	1910.7094

TABLE 4.7: Results of hyperheuristic experiments on TTP NL12

TTP NL12		Initial Solution Quality				167249082
Algorithm		Feasible	Worst	Average	Best	Std Dev
Simple Random						
	AM	0	N/A	N/A	N/A	N/A
	OI	1	144821	140142.2	133971	2901.7612
Ant Algorithm						
Walk 1	AM	1	160721	157864.2	152670	1993.1797
Walk 2	AM	1	158541	156242.4	153635	1318.5318
Walk 3	AM	1	159236	156084.44	153843	1441.4126
Walk 4	AM	1	159049	156760.68	152946	1369.3619
Walk 5	AM	1	160472	157357.	152909	2074.8852
Walk 6	AM	1	160616	157727.56	153612	2071.5457
Walk 8	AM	1	161347	157935.68	151735	2467.7512
Walk 10	AM	1	163183	158600.88	154358	2093.9785
Walk 12	AM	1	162303	157606.76	152306	2659.6714
Walk 15	AM	1	163849	159371.36	155344	2325.8013
Walk 1	OI	1	146630	139251.44	134774	2863.1470
Walk 2	OI	1	147061	139637.96	134336	3183.1573
Walk 3	OI	1	**142344**	138548.52	134429	2366.4988
Walk 4	OI	1	148479	139167.04	131119	3274.1683
Walk 5	OI	1	144252	138678.52	132967	3674.3645
Walk 6	OI	1	143802	**138417.16**	132888	2910.0837
Walk 8	OI	1	145831	138708.92	132840	3584.3257
Walk 10	OI	1	143760	140214.76	135529	2208.0302
Walk 12	OI	1	142494	138506.04	**130882**	2976.8130
Walk 15	OI	1	144524	138785.88	133113	3051.4636
Ants 1	AM	0	N/A	N/A	N/A	N/A
Ants 2	AM	0.16	171488	168276.	166142	2305.6416
Ants 3	AM	0.84	170518	164881.43	157580	3613.3551
Ants 4	AM	1	164495	159903.96	156491	1747.5693
Ants 5	AM	1	160472	157357.	152909	2074.8852
Ants 6	AM	1	158210	155279.04	150988	1944.6478
Ants 8	AM	1	155828	152892.68	148559	1717.3584
Ants 10	AM	1	155149	151441.68	148283	1573.6915
Ants 12	AM	1	152201	149359.2	144314	1730.4543
Ants 15	AM	1	149643	147006.52	143758	1646.5030
Ants 1	OI	1	146778	139879.28	133900	3257.8374
Ants 2	OI	1	145335	139368.24	133451	3064.0604
Ants 3	OI	1	146228	139000.56	133627	3231.2777
Ants 4	OI	1	145466	139482.08	133650	2980.9798
Ants 5	OI	1	144252	138678.52	132967	3674.3645
Ants 6	OI	1	144624	139155.08	133368	2881.0030
Ants 8	OI	1	144597	138814.64	133314	2725.8786
Ants 10	OI	1	146226	139072.2	133987	2715.8464
Ants 12	OI	1	144786	139465.4	134473	2652.3808
Ants 15	OI	1	143150	139020.48	134073	2486.4675

TABLE 4.8: Results of hyperheuristic experiments on TTP NL14

TTP NL14		Initial Solution Quality				1119914601
Algorithm		Feasible	Worst	Average	Best	Std Dev
Simple Random						
	AM	0	N/A	N/A	N/A	N/A
	OI	1	269110	257624.04	240682	6145.0393
Ant Algorithm						
Walk 1	AM	0.88	329540	312675.55	297199	7990.8335
Walk 2	AM	1	321097	308688.24	299710	4948.4843
Walk 3	AM	0.92	321148	307239.83	295465	5846.6848
Walk 4	AM	0.88	322563	310875.5	302231	5946.5912
Walk 5	AM	0.96	327442	311170.17	297808	6663.6552
Walk 6	AM	0.84	321105	310955.33	302521	4953.0762
Walk 8	AM	0.84	319789	311672.81	302571	4707.1812
Walk 10	AM	0.6	325199	310111.93	298132	7634.9510
Walk 12	AM	0.68	328056	313715.41	297267	8070.8762
Walk 15	AM	0.52	334736	314578.92	300959	9413.9231
Walk 1	OI	1	268793	254862.2	241725	6331.9080
Walk 2	OI	1	267338	257314.72	243799	5419.4586
Walk 3	OI	1	273663	259026.24	245097	6692.8835
Walk 4	OI	1	268897	255737.48	248115	5230.1004
Walk 5	OI	1	**262681**	255873.16	244899	4624.9625
Walk 6	OI	1	265322	256470.04	247415	5268.5600
Walk 8	OI	1	269456	257231.36	243732	6694.7915
Walk 10	OI	1	267115	258187.04	245384	5390.0410
Walk 12	OI	1	267935	256747.36	244833	5694.0958
Walk 15	OI	1	275047	258472.88	248274	7008.8409
Ants 1	AM	0	N/A	N/A	N/A	N/A
Ants 2	AM	0	N/A	N/A	N/A	N/A
Ants 3	AM	0.04	315147	315147.	315147	0.
Ants 4	AM	0.4	323916	314374.2	308499	4926.0399
Ants 5	AM	0.96	327442	311170.17	297808	6663.6552
Ants 6	AM	1	314173	303469.48	293544	5754.3674
Ants 8	AM	1	303214	295779.04	288797	4299.8869
Ants 10	AM	1	300707	292246.64	281884	3963.0758
Ants 12	AM	1	293063	288161.56	284608	2317.0944
Ants 15	AM	1	289398	284169.92	274064	3309.9727
Ants 1	OI	1	268188	256387.28	244933	5102.6474
Ants 2	OI	1	272436	258657.24	247986	5695.1191
Ants 3	OI	1	265604	256585.04	250861	4103.1104
Ants 4	OI	1	266821	256285.52	**240420**	6837.4183
Ants 5	OI	1	**262681**	255873.16	244899	4624.9625
Ants 6	OI	1	267337	256445.12	242379	5424.9703
Ants 8	OI	1	269558	256110.52	244421	6406.7977
Ants 10	OI	1	269383	257691.08	248553	5523.0908
Ants 12	OI	1	269668	255773.56	243812	6336.7079
Ants 15	OI	1	263088	**254581.48**	243301	4944.9591

TABLE 4.9: Results of hyperheuristic experiments on TTP NL16

TTP NL16		Initial Solution Quality				1221354386
Algorithm		Feasible	Worst	Average	Best	Std Dev
Simple Random						
	AM	0	N/A	N/A	N/A	N/A
	OI	1	375219	360151.36	347867	7522.4159
Ant Algorithm						
Walk 1	AM	0.08	453696	449849.	446002	5440.4796
Walk 2	AM	0.2	469496	454039.2	445269	9480.6150
Walk 3	AM	0.16	469819	461689.25	450980	7892.8782
Walk 4	AM	0.2	479047	458948.8	449390	11893.9033
Walk 5	AM	0.16	460596	452720.75	436083	11485.0679
Walk 6	AM	0.16	460684	452629.5	443601	7006.1469
Walk 8	AM	0.04	460593	460593.	460593	0.
Walk 10	AM	0	N/A	N/A	N/A	N/A
Walk 12	AM	0.04	465525	465525.	465525	0.
Walk 15	AM	0.08	453636	448560.5	443485	7177.8409
Walk 1	OI	1	379950	359985.24	347806	7619.6647
Walk 2	OI	1	380596	358714.12	347282	8350.9690
Walk 3	OI	1	**365361**	356255.76	344383	6195.7061
Walk 4	OI	1	382754	359725.32	336559	9949.1627
Walk 5	OI	1	375577	357308.08	345735	7526.1796
Walk 6	OI	1	369571	355987.4	343016	6204.9955
Walk 8	OI	1	389185	362115.68	345895	8863.5678
Walk 10	OI	1	381162	357676.32	343614	9056.1298
Walk 12	OI	1	374744	360188.16	344976	7156.3800
Walk 15	OI	1	369129	356472.52	339960	9137.9080
Ants 1	AM	0	N/A	N/A	N/A	N/A
Ants 2	AM	0	N/A	N/A	N/A	N/A
Ants 3	AM	0	N/A	N/A	N/A	N/A
Ants 4	AM	0	N/A	N/A	N/A	N/A
Ants 5	AM	0.16	460596	452720.75	436083	11485.0679
Ants 6	AM	0.32	463674	450990.94	441726	6260.1382
Ants 8	AM	0.96	452532	437044.38	418541	7224.3063
Ants 10	AM	1	440205	427446.28	414560	6321.8593
Ants 12	AM	1	427413	420330.32	410945	3934.5627
Ants 15	AM	1	422898	412065.48	405400	4903.7795
Ants 1	OI	1	367191	356433.	341186	7961.3168
Ants 2	OI	1	368269	357106.4	345306	7608.7133
Ants 3	OI	1	373484	358680.56	338989	7825.1795
Ants 4	OI	1	372243	356575.44	340575	8182.2554
Ants 5	OI	1	375577	357308.08	345735	7526.1796
Ants 6	OI	1	367063	**355508.68**	**335959**	8492.8521
Ants 8	OI	1	375775	357903.4	343265	8436.6125
Ants 10	OI	1	389207	359623.04	342688	9349.3469
Ants 12	OI	1	369670	357023.8	340097	7837.6556
Ants 15	OI	1	366642	356533.52	343522	5979.5856

Approach	*csit0*	*csit1*	*csit2*	Time
Soubeiga, Random HH [Sou03]	-1193.19	-2880.93	-1614.11	600
Soubeiga, Choice Function HH [Sou03]	-1444.99	-2963.37	-1724.15	600
Han and Kendall, HyperGA [HK03b]	-1453.32			924
O'Brien, Random HH	-1628.31	-3014.97	-1805.61	24-13
O'Brien, Ant Algorithm HH	**-1651.09**	**-3018.64**	**-1832.47**	24

TABLE 4.10: Averaged results of approaches to the Project Presentation Scheduling Problem. Results from other papers are above the horizontal dividing line; results from this thesis are below that line.

Approach	*NL6*	*NL8*	*NL10*	*NL12*	*NL14*	*NL16*
Cardemil		40416	66037	118955	205894	308413
Dorrepaal			66369	119990	189759	
Easton	**23916**	**39721**				312623
Langford			**59436**	112298	190056	272902
Lanchi, Lapierre and Laporte				138850	262010	
Rottembourg and Laburthe			68691	143655	301113	
Shen						281660
Trick	23978					
Van Hentenryck			59583	**110729**	**188728**	**261687**
Zhang		39947	61608	119102	207075	293175
Chen	**23916**	40391	65168	123752	225169	321037
O'Brien, Ant Algorithm HH	**23916**	41927	69232	130882	240420	335959

TABLE 4.11: Best results of approaches to the Travelling Tournament Problem, available at [Tri07], with hyperheuristic approaches from Chen [Che07]. Results from other papers are above the first horizontal dividing line, followed by other hyperheuristic results above the second line, followed by results from this thesis. *"Van Hentenryck" is Anagnostopoulos, Michel, Van Hentenryck and Vergados*

that being able to explore poor moves enables the hyperheuristics to avoid local optima.

However, for larger instances the Any Moves hyperheuristics struggle to reach feasible solutions, and we note that while the journey length has no apparent effect on whether the hyperheuristic finds feasible solutions, a large colony seems to be better at finding feasible solutions than a small colony. Since that the Any Moves Ant Algorithm always relocates to the best solution found during the previous cycle (not the best solution

found so far in the whole search), the hyperheuristic is still capable of choosing continually poorer solutions, or cycling between degrees of quality. A greater number of ants during each given cycle increases the chance of one of them maintaining a solution of similar quality to the current solution, whereas a greater path length only increases the likelihood of an ant finding a different solution regardless of how many ants there are.

This hypothesis may be reinforced by the single instance in which the hyperheuristics presented here discovered the optimal solution to the *NL6* instance: the Ant Algorithm Hyperheuristic variant which accepted any move and which employed 12 ants.

The Only Improving Hyperheuristics, however, find feasible solutions every time. This does suggest that finding feasible solutions may not be complicated if the heuristics choose the correct combinations of teams and timeslots, but that the randomness of all heuristics used for this problem work against the hyperheuristic (a theory perhaps reinforced by the strong performance of Chen's version of the hyperheuristic, which employed five additional greedy heuristics which always move to better solutions where a better solution is available).

For the Project Presentation Scheduling Problem, trends are more difficult to identify. The Any Moves criterion appears to serve just as well in the csit0 instance, suggesting that good moves are not that difficult to find during each cycle, whereas the Only Improving criterion performs better in the more difficult instances. With the exception of Table 4.1, the better results in each category all occur to hyperheuristics which use short journey lengths, which may indicate that longer journey lengths are unnecessarily long.

We examine the best runs from each instance to further determine the Ant Algo-

rithm Hyperheuristic's behaviour, including any evidence of its learning. For each instance, we present six images. In the first image (a), the solution quality of all accepted moves are presented in green with their respective improvements upon the prior solution marked in red (for the Travelling Tournament Problem images this quality only represents the distance aspect of the solution, not how feasible it is).

In the second and third images (b, c), we attempt to present several ongoing statistics about the hyperheuristic's selection mechanism. Each mark represents a cycle of the run. "Acceptence", "Improvement" and "Absolute" indicates what fraction of the moves made during each cycle respectively were accepted, produced an improvement, and produced an absolute improvement (i.e. found a new best overall solution). "Convergence" indicates how much the hyperheuristic favours some heuristics more than others, by totalling the proportions used above that which would be expected by using each equally and scaling to the range 0...1: thus, a hyperheuristic which calls every heuristic uniformly during a cycle will maintain a convergence of 0, and a hyperheuristic which uses only one heuristic every cycle will maintain a convergence of 1. "Confidence" measures the average selection of heuristics, by scaling between 0...1 the probability of selecting a heuristic between the probabilities of selecting the currently-believed worst and best heuristics, and averaging the selection over a cycle.

The fourth, fifth and sixth images (d, e, f) display the proportion of heuristic calls each heuristic received per step, per cycle and per prior hundred moves (in the latter case, again, marked at the first and last hundred moves and each absolute improvement).

The *NL6* graphs (Figures 4.7-4.12) are understandably erratic. Since this is the

only problem instance of the nine in which the best result was produced by an "Any Moves" hyperheuristic, the Improvement/Quality graph (Figure 4.7) shows no clear route from the initial solution to the best solution. The proportion of acceptances in Figure 4.8 is naturally close to 1 throughout[4]. It can be seen that many of the heuristic calls produced an improvement (12,083 of them), and while very few make an absolute improvement it does appear that the flexibility of the Any Moves criterion is important: the best solution is the result of the 32,668th heuristic call, a total of 705 heuristic moves from the initial solution.

[4]The reason that the acceptance is not *exactly* 1 throughout is because the Any Moves criterion does actually reject some solutions: those which are exactly the same. The principle is to not reward a heuristic which makes no difference whatsoever.

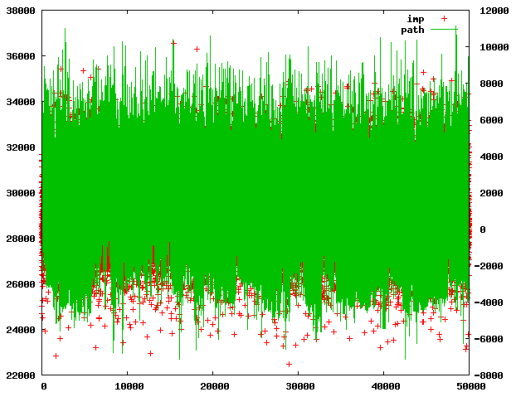

FIGURE 4.7: Instance *NL6*(a): Moves accepted: Solution Quality and Improvement

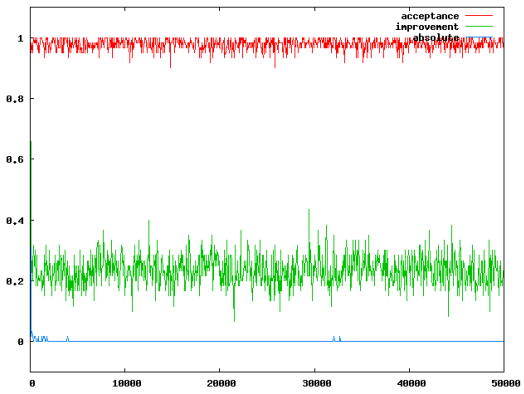

FIGURE 4.8: Instance *NL6*(b): Moves Accepted, Improving and Improving Absolutely

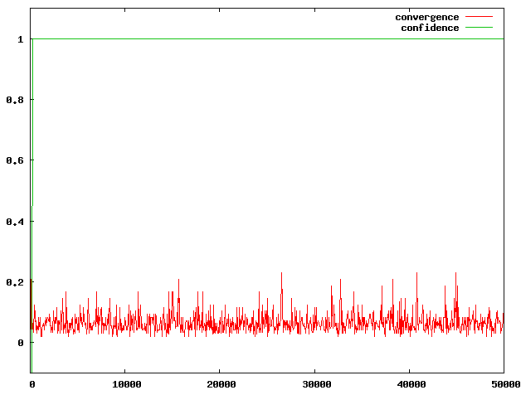

FIGURE 4.9: Instance *NL6*(c): Confidence and Convergence

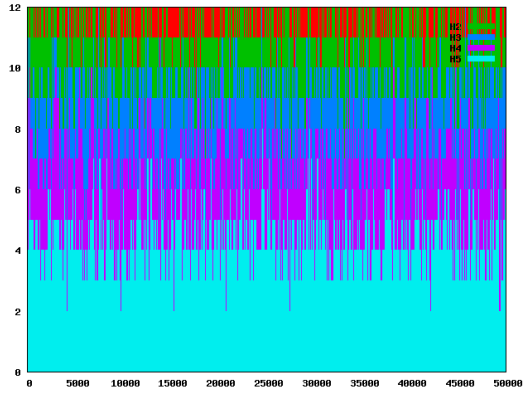

FIGURE 4.10: Instance *NL6*(d): Proportion of Heuristic Calls per Heuristic per Step

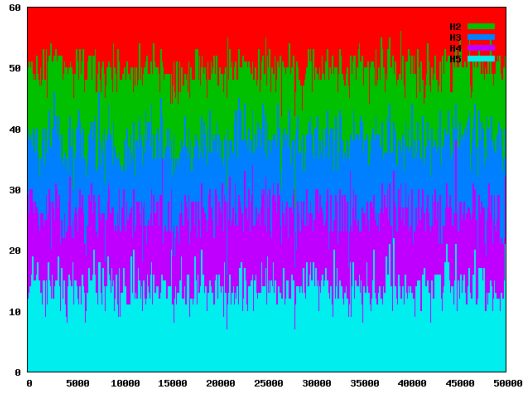

FIGURE 4.11: Instance *NL6*(e): Proportion of Heuristic Calls per Heuristic per Cycle

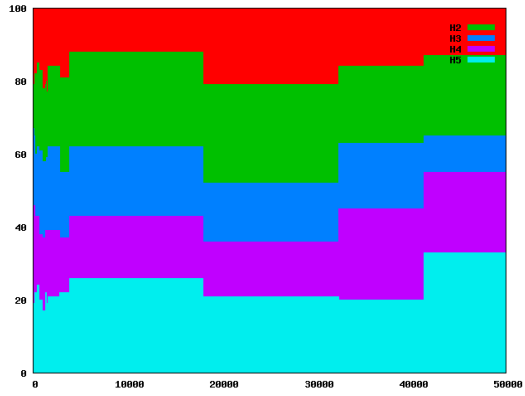

FIGURE 4.12: Instance *NL6*(f): Proportion of Heuristic Calls per Heuristic in the 100 Calls prior to each Absolute Improvement

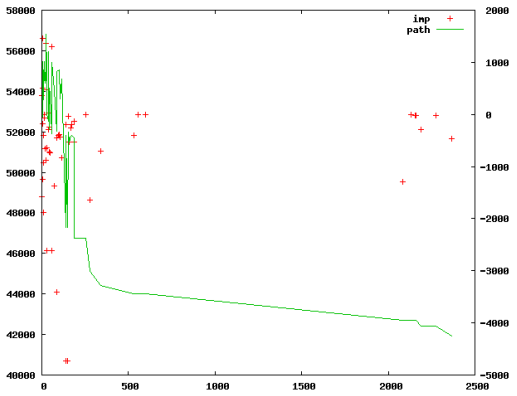

FIGURE 4.13: Instance *NL8*(a): Moves accepted: Solution Quality and Improvement

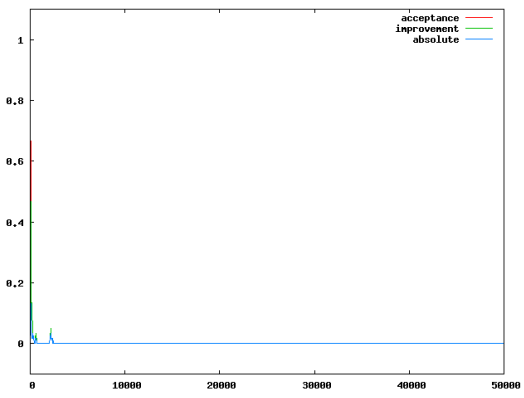

FIGURE 4.14: Instance *NL8*(b): Moves Accepted, Improving and Improving Absolutely

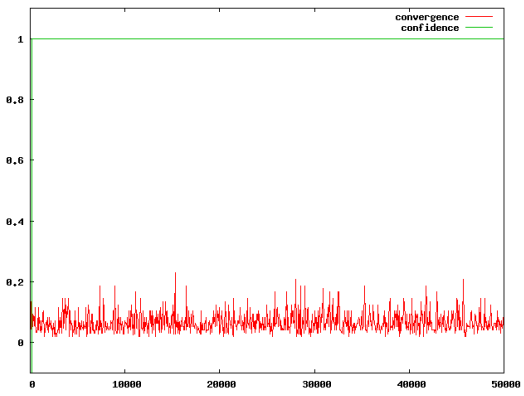

FIGURE 4.15: Instance *NL8*(c): Confidence and Convergence

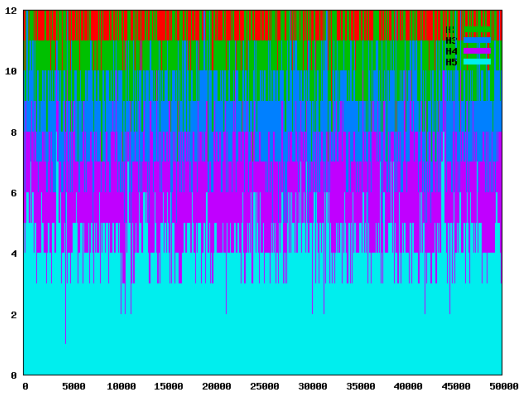

FIGURE 4.16: Instance *NL8*(d): Proportion of Heuristic Calls per Heuristic per Step

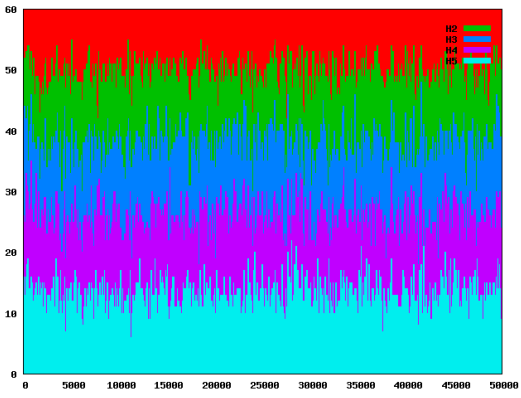

FIGURE 4.17: Instance *NL8*(e): Proportion of Heuristic Calls per Heuristic per Cycle

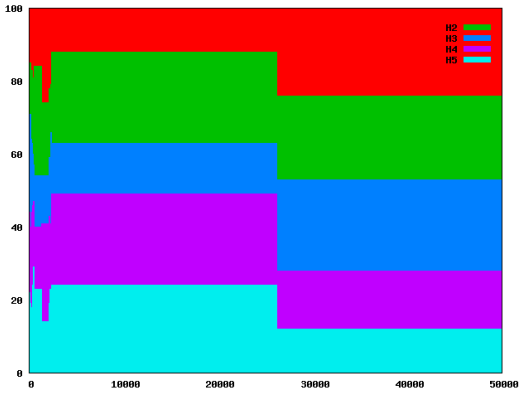

FIGURE 4.18: Instance *NL8*(f): Proportion of Heuristic Calls per Heuristic in the 100 Calls prior to each Absolute Improvement

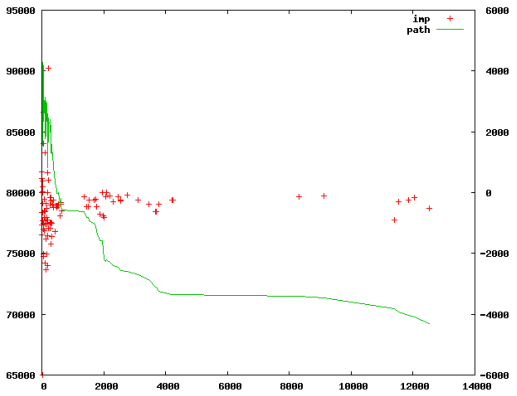

FIGURE 4.19: Instance *NL10*(a): Moves accepted: Solution Quality and Improvement

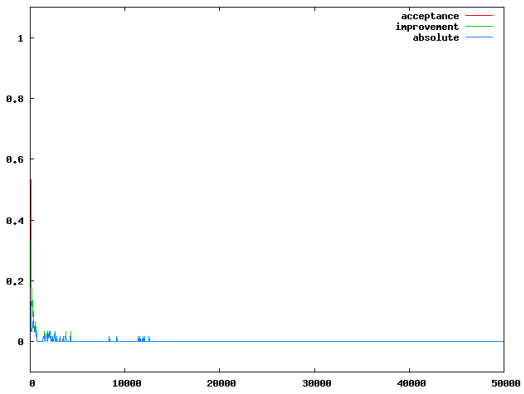

FIGURE 4.20: Instance *NL10*(b): Moves Accepted, Improving and Improving Absolutely

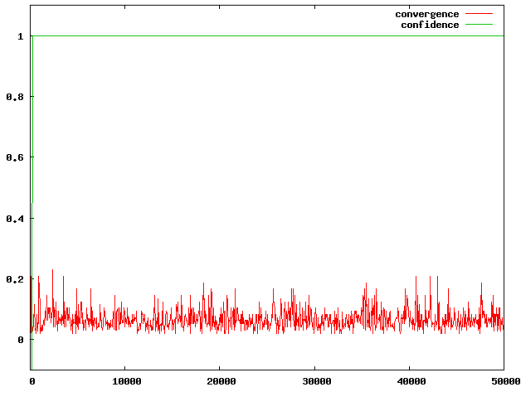

FIGURE 4.21: Instance *NL10*(c): Confidence and Convergence

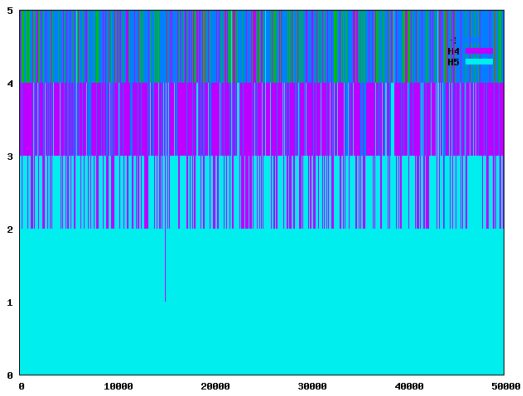

FIGURE 4.22: Instance *NL10*(d): Proportion of Heuristic Calls per Heuristic per Step

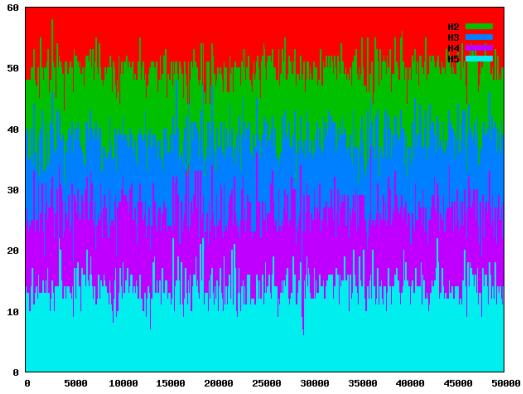

FIGURE 4.23: Instance *NL10*(e): Proportion of Heuristic Calls per Heuristic per Cycle

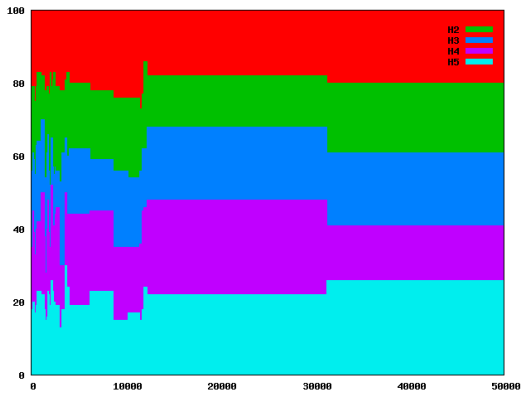

FIGURE 4.24: Instance *NL10*(f): Proportion of Heuristic Calls per Heuristic in the 100 Calls prior to each Absolute Improvement

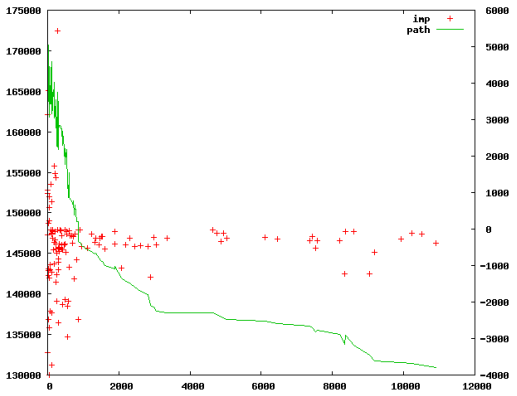

FIGURE 4.25: Instance *NL12*(a): Moves accepted: Solution Quality and Improvement

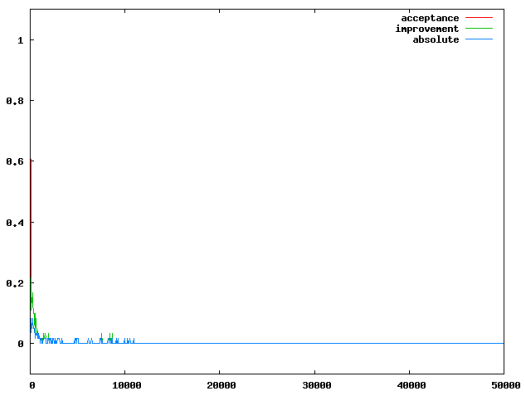

FIGURE 4.26: Instance *NL12*(b): Moves Accepted, Improving and Improving Absolutely

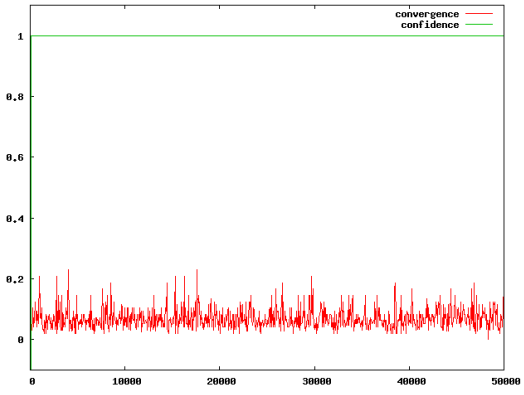

FIGURE 4.27: Instance *NL12*(c): Confidence and Convergence

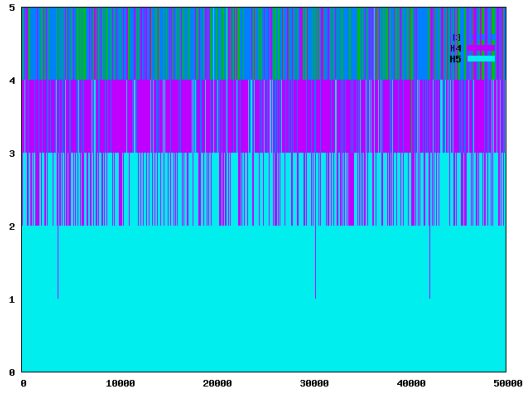

FIGURE 4.28: Instance *NL12*(d): Proportion of Heuristic Calls per Heuristic per Step

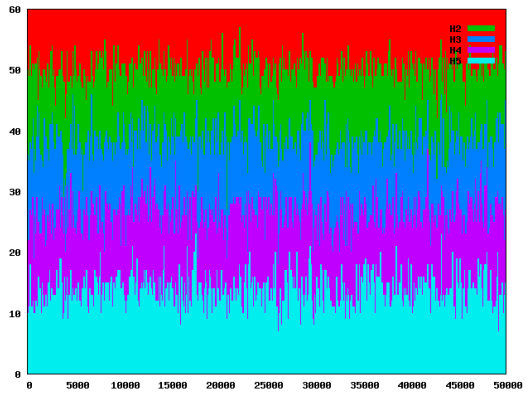

FIGURE 4.29: Instance *NL12*(e): Proportion of Heuristic Calls per Heuristic per Cycle

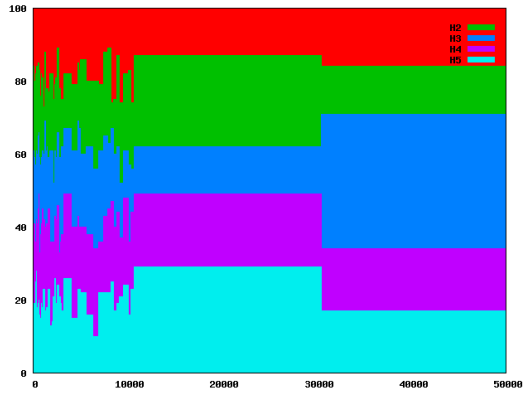

FIGURE 4.30: Instance *NL12*(f): Proportion of Heuristic Calls per Heuristic in the 100 Calls prior to each Absolute Improvement

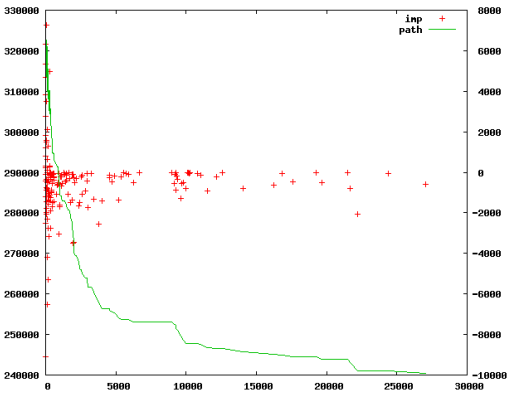

FIGURE 4.31: Instance *NL14*(a): Moves accepted: Solution Quality and Improvement

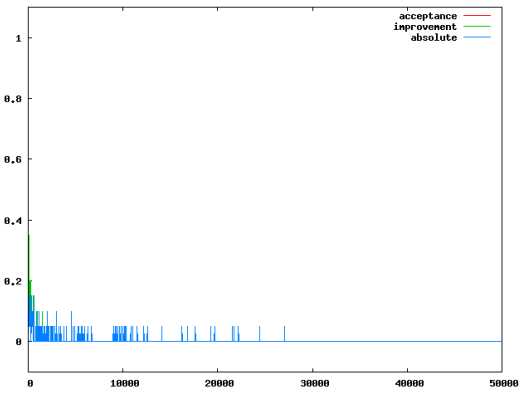

FIGURE 4.32: Instance *NL14*(b): Moves Accepted, Improving and Improving Absolutely

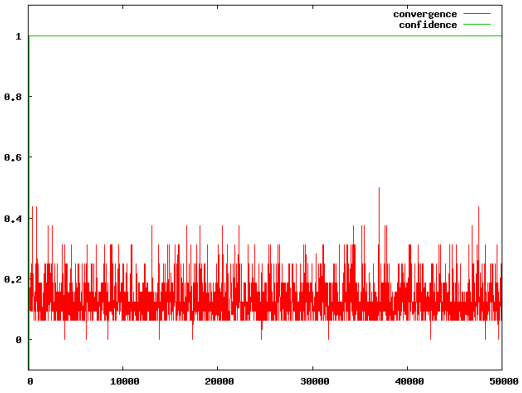

FIGURE 4.33: Instance *NL14*(c): Confidence and Convergence

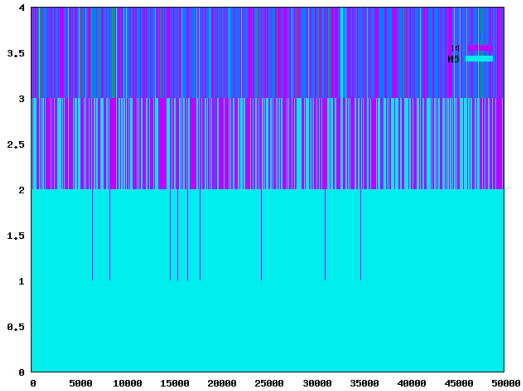

FIGURE 4.34: Instance *NL14*(d): Proportion of Heuristic Calls per Heuristic per Step

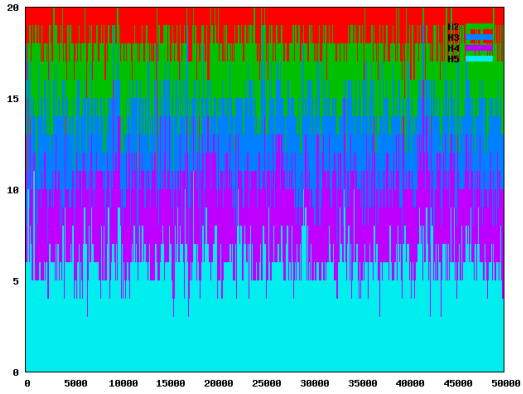

FIGURE 4.35: Instance *NL14*(e): Proportion of Heuristic Calls per Heuristic per Cycle

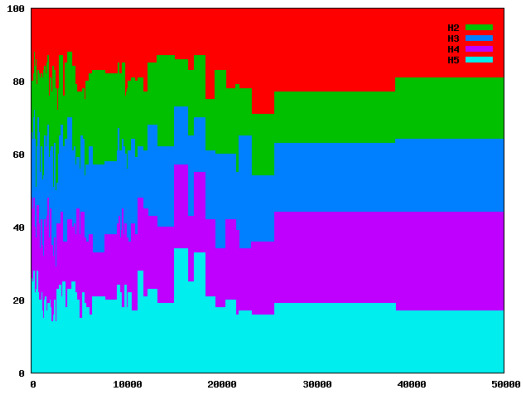

FIGURE 4.36: Instance *NL14*(f): Proportion of Heuristic Calls per Heuristic in the 100 Calls prior to each Absolute Improvement

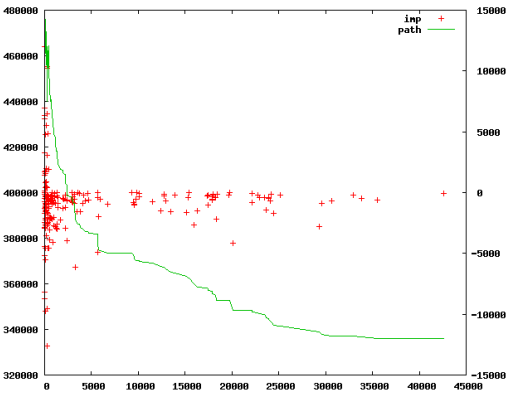

FIGURE 4.37: Instance *NL16*(a): Moves accepted: Solution Quality and Improvement

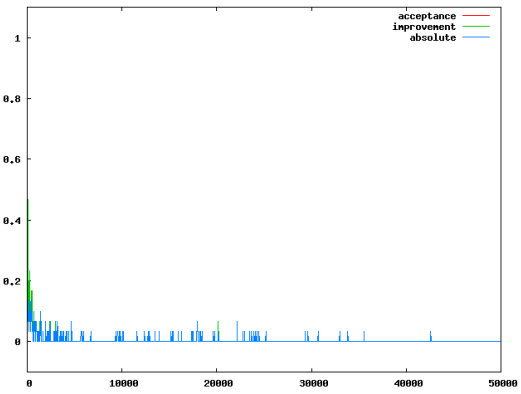

FIGURE 4.38: Instance *NL16*(b): Moves Accepted, Improving and Improving Absolutely

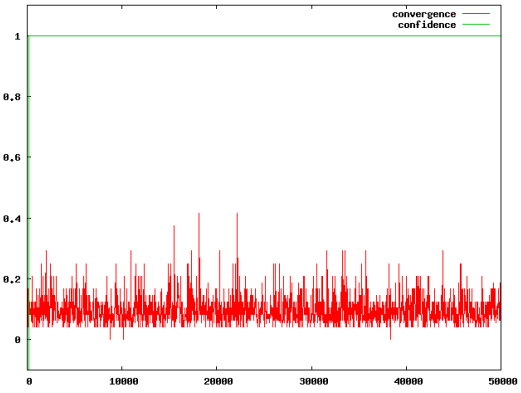

FIGURE 4.39: Instance *NL16*(c): Confidence and Convergence

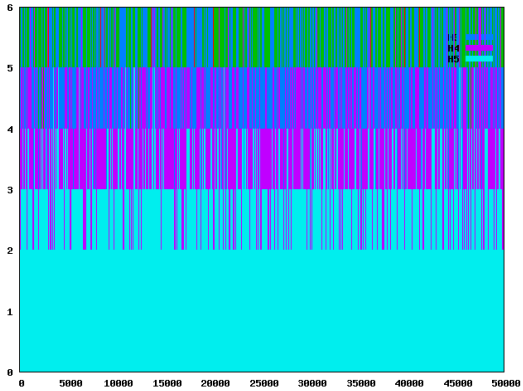

FIGURE 4.40: Instance *NL16*(d): Proportion of Heuristic Calls per Heuristic per Step

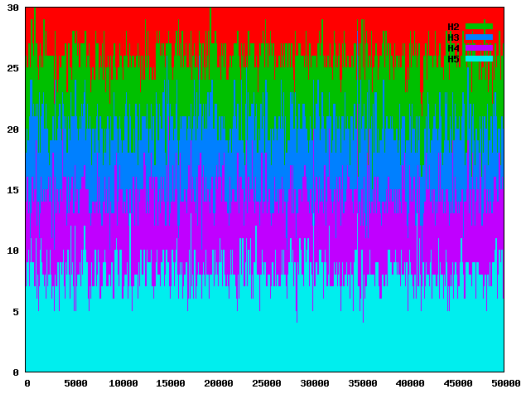

FIGURE 4.41: Instance *NL16*(e): Proportion of Heuristic Calls per Heuristic per Cycle

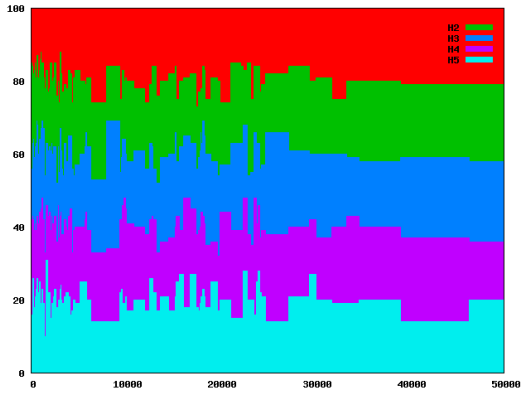

FIGURE 4.42: Instance *NL16*(f): Proportion of Heuristic Calls per Heuristic in the 100 Calls prior to each Absolute Improvement

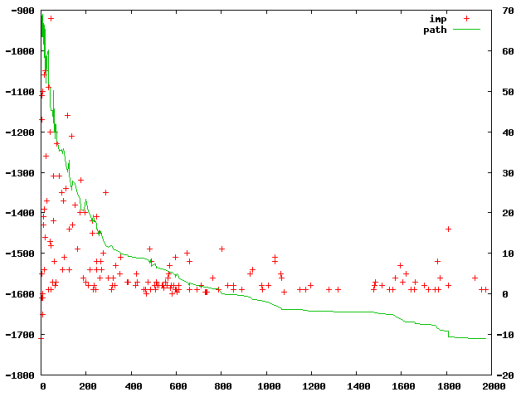

FIGURE 4.43: Instance *csit0*(a): Moves accepted: Solution Quality and Improvement

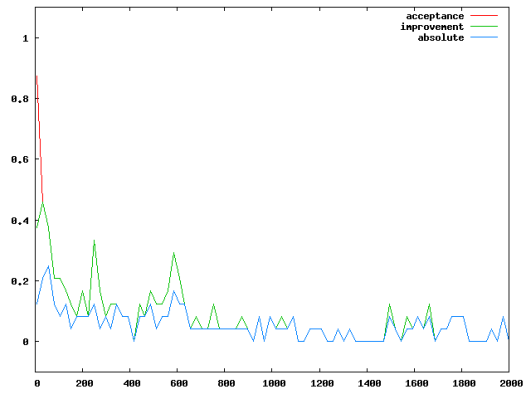

FIGURE 4.44: Instance *csit0*(b): Moves Accepted, Improving and Improving Absolutely

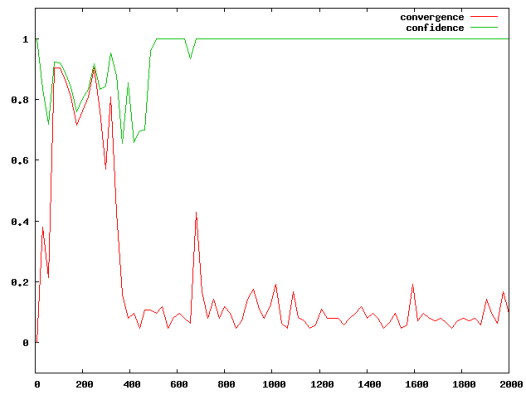

FIGURE 4.45: Instance *csit0*(c): Confidence and Convergence

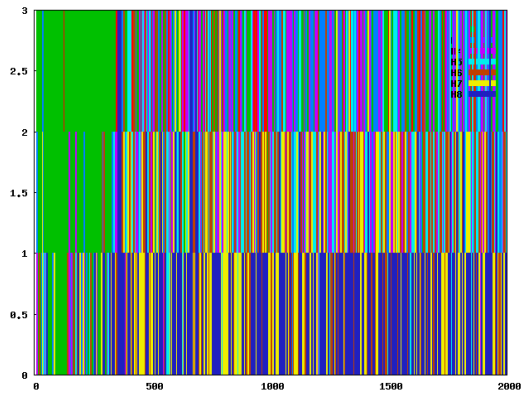

FIGURE 4.46: Instance *csit0*(d): Proportion of Heuristic Calls per Heuristic per Step

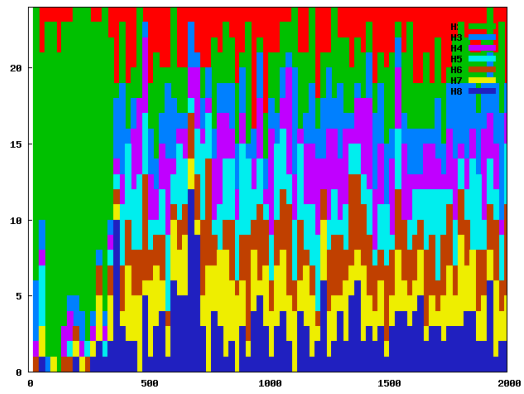

FIGURE 4.47: Instance *csit0*(e): Proportion of Heuristic Calls per Heuristic per Cycle

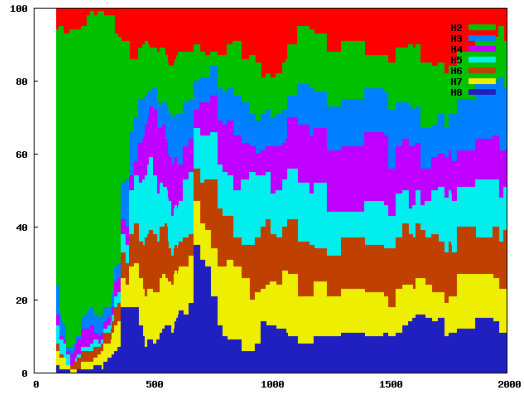

FIGURE 4.48: Instance *csit0*(f): Proportion of Heuristic Calls per Heuristic in the 100 Calls prior to each Absolute Improvement

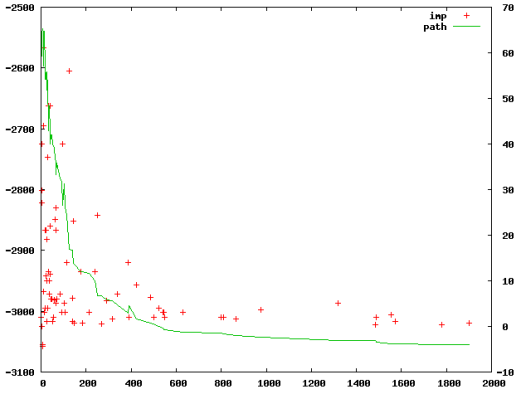

FIGURE 4.49: Instance *csit1* (a): Moves accepted: Solution Quality and Improvement

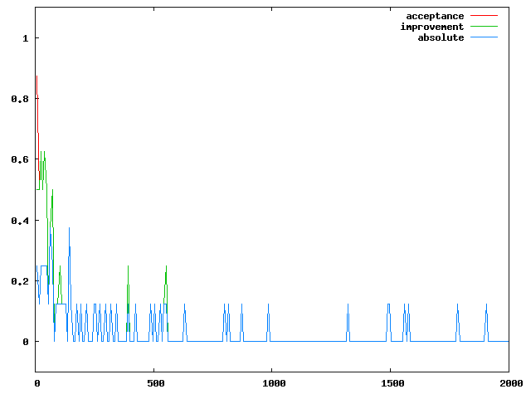

FIGURE 4.50: Instance *csit1* (b): Moves Accepted, Improving and Improving Absolutely

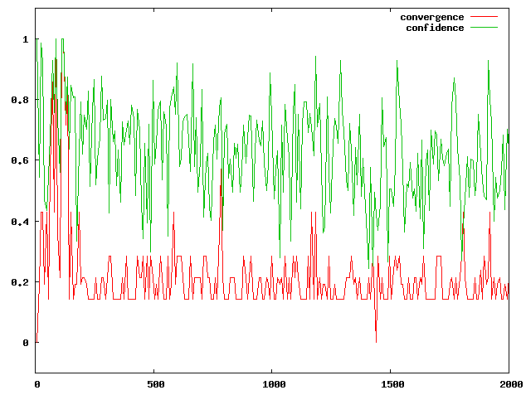

FIGURE 4.51: Instance *csit1* (c): Confidence and Convergence

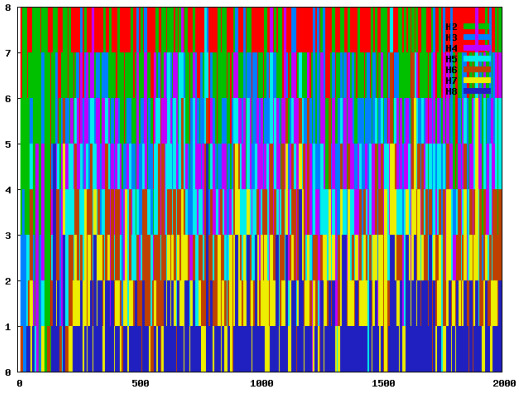

FIGURE 4.52: Instance *csit1* (d): Proportion of Heuristic Calls per Heuristic per Step

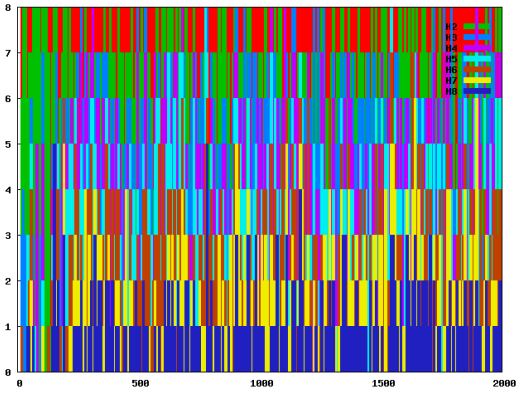

FIGURE 4.53: Instance *csit1* (e): Proportion of Heuristic Calls per Heuristic per Cycle

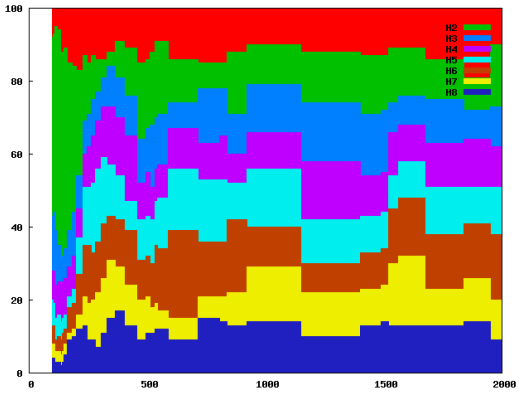

FIGURE 4.54: Instance *csit1* (f): Proportion of Heuristic Calls per Heuristic in the 100 Calls prior to each Absolute Improvement

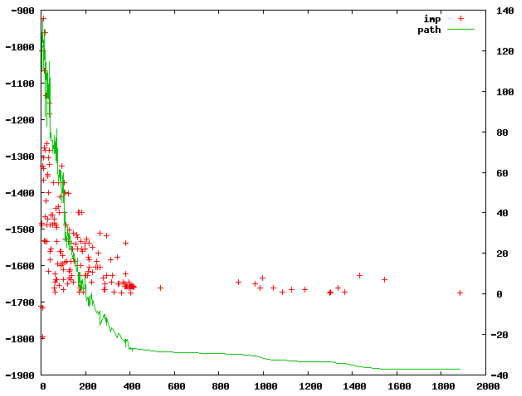

FIGURE 4.55: Instance *csit2*(a): Moves accepted: Solution Quality and Improvement

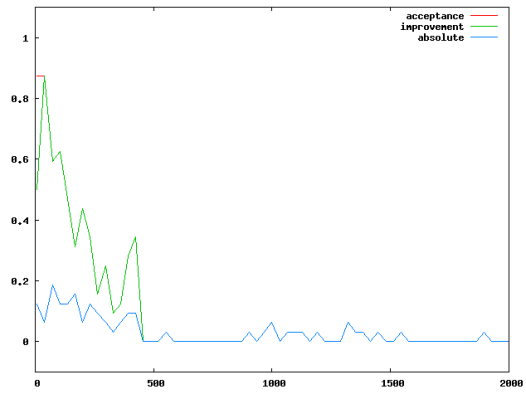

FIGURE 4.56: Instance *csit2*(b): Moves Accepted, Improving and Improving Absolutely

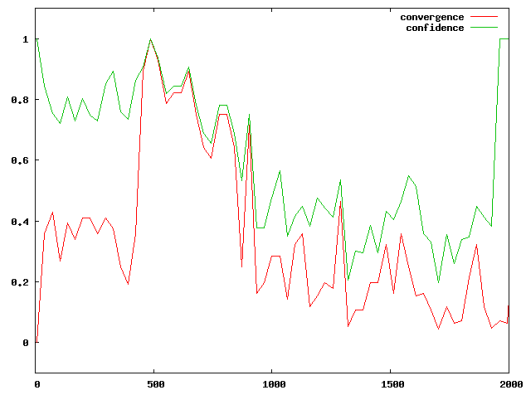

FIGURE 4.57: Instance *csit2*(c): Confidence and Convergence

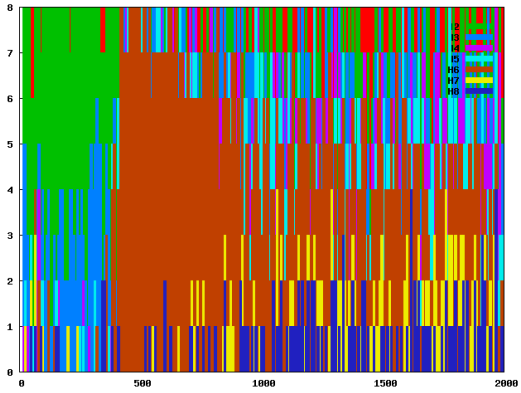

FIGURE 4.58: Instance *csit2*(d): Proportion of Heuristic Calls per Heuristic per Step

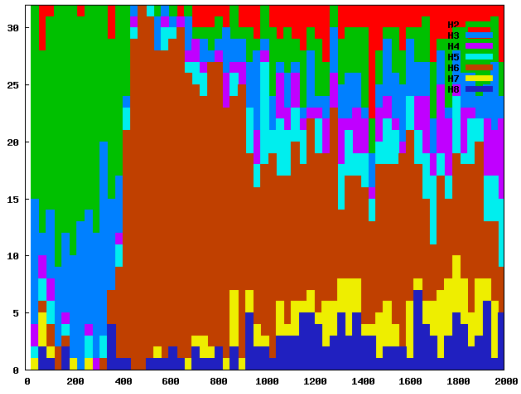

FIGURE 4.59: Instance *csit2*(e): Proportion of Heuristic Calls per Heuristic per Cycle

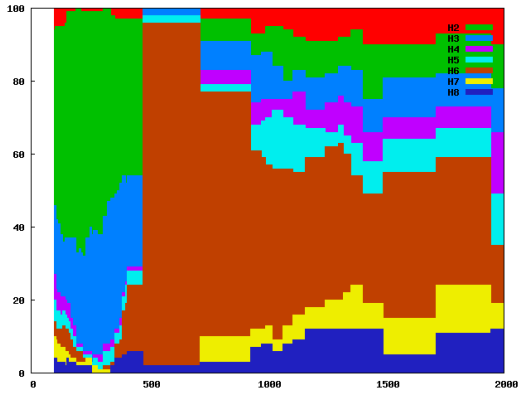

FIGURE 4.60: Instance *csit2*(f): Proportion of Heuristic Calls per Heuristic in the 100 Calls prior to each Absolute Improvement

The Travelling Tournament Problem instances seem to have many similarities in their best runs. The portion of Absolute Improvement moves is high early in each run, as the hyperheuristic works through the feasible area of the search space. The "Confidence" for each run is 1, virtually throughout, indicating that the hyperheuristic is always choosing the heuristic it thinks is best, while the "Convergence" remains low, indicating that the heuristics are chosen with almost equal probability; this is reinforced by figures (d) and (e), which indicate that heuristic Th_5 is favoured most. The 'best heuristic' therefore changes from step to step, and the visibility and pheromone components of the hyperheuristic are therefore very erratic, drastically changing the probabilities for each heuristic from step to step. While it appears that the hyperheuristic is unstable, it should also be noted that the heuristics are unstable, always choosing random moves rather than moves designed to improve a solution.

The Project Presentation Scheduling Problem instances are more interesting, as befits a set of heuristics where some heuristics are designed to improve. We observe that for instance *csit0*, the Confidence in Figure 4.45 is initially 1, varies a great deal in the early part of the search, and returns to 1; in the later instances (Figures 4.51, 4.57) the Confidence is more erratic, indicating that the probability distribution across the set of heuristics is more fair. As these instances are all Only Improving variations, it is not surprising that all accepted moves (indicated by the green line) are improvements (and therefore subsumed by the dark blue); however, as the instances progress, we observe fewer improving moves which are not absolute improvements as the searches progress, reinforcing the theory that the instances progress in difficulty.

We also note several significant convergences: in the first 500 heuristic calls, heuristic Ph_3 is implemented most often; beyond that point, for instances *csit0* and *csit1*, this early convergence fades and other heuristics are called more prominently. For instance *csit2*, however, the hyperheuristic converges upon heuristic Ph_6. We note that both of these heuristics are the most specialised of their type in the heuristic set, intended to produce better solutions.

4.6 Conclusions

We have created a hyperheuristic algorithm by hybridising the ant algorithm technique with the choice function hyperheuristic and applied it to two problems. The results of our experiments are superior to the Project Presentation Scheduling Problem published in [CKS02, KSC02, HK02, Sou03] (see Table 4.10) and competitive with prior experiments for the Travelling Tournament Problem (see Table 4.11). In both cases the hyperheuristic produces results in less running time than previous efforts, though it is unclear how much credit should be given to the hyperheuristic rather than the constructive heuristic or the initial solution.

We have analysed the progress of the hyperheuristic during its experiments by creating and analysing several measurements which assess how much each selected heuristic is favoured at the decision point. We have observed that the ant algorithm hyperheuristic can identify certain heuristics' performances as being good and then converge upon those heuristics. We have observed that the ant algorithm hyperheuristic can also identify when those same heuristics' performances worsen (relative to those of other heuristics) and

then diverge from them. We conclude that the ant algorithm hyperheuristic learns as it progresses.

However, we also observe that the visibility and pheromone combination is volatile, making it difficult to identify the contribution of each specific component so the selection mechanism of the hyperheuristic can be better understood and perhaps improved.

We have tested this ant algorithm hyperheuristic by experimentation with variables: the number of ants in the colony, the number of heuristic calls each ant must explore in each cycle, and the criterion by which the ant chooses to accept or reject a discovered solution. We observe that the acceptance criterion is generally more significant to the performance of the hyperheuristic than the other variables. An interesting future research direction could be to create an ant algorithm hyperheuristic in which the colony can be divided between ants using different acceptance criteria.

However, we note that it may be the structure of the hyperheuristic technique - the mechanism by which multiple paths of heuristic calls are explored and analysed, and the mechanism which returns the colony to the best solution of that cycle for further exploration - which is the most signficant part of the hyperheuristic. Efforts have been made in the literature [BOK06] to explore the effect of the heuristic selection mechanism and acceptance criterion of hyperheuristics, but more effort could be devoted to exploring the effect of different mechanisms in hyperheuristic structures: the metaheuristic component.

Chapter 5

Conclusions and Future Work

> "The data about Earth speaks for itself–" Selv's thin, angry voice came back.
> "*No* data speaks for itself," McCoy said, forceful. "Data just lies there. *People* speak."
>
> –*Star Trek: Spock's World*, by Diane Duane

The contributions of this thesis are as follows:

- We have presented a new hyperheuristic technique based upon the ant algorithm metaheuristic.
- We have demonstrated the generality of the technique by applying it to two distinct problem classes.
- We have demonstrated the capabilities of the hyperheuristic framework by producing competitive or superior results to those previously published in the literature for both real-world problems, though the ant algorithm hyperheuristic did not significantly outperform a random hyperheuristic.

We have considered a range of values considered to possibly effect the performance of the hyperheuristic. These were shown to have little effect upon the hyperheuristic's

results, in the case of these two problem classes considered. There may be some merit to developing an adaptive variant of the hyperheuristic to dynamically alter these values depending on how 'difficult' the solution space seems to be at a given point.

A more promising direction may be to pursue lines of work presented by Cowling and Chaklevitch [CC03, CC05]. A large set of heuristics may be available to the hyperheuristic, restricted to a 'promising' subset of heuristics during each cycle. Other means of strategising the hyperheuristic's choice of heuristics at the beginning of a cycle may be considered, as the heuristic selection mechanism we use does not seem particularly stable.

Combining the method with a more relaxed acceptance criterion such as simulated annealing may also produce better results. A number of probabilistic acceptance criteria were experimented with but not considered in depth, as the temperature function of the simulated annealing technique would also involve some domain-specific knowledge and tuning. One possibility is to diversify the roles of the ants by assigning different acceptance criteria to different ants.

An investigation of aspects of the hyperheuristic algorithms which do not include heuristic selection or acceptance criteria may be very interesting. The decision point at which the ant algorithm hyperheuristic branches into considering multiple heuristic sequences are presently arbitrary, determined only by the number of calls made until that point, and the best solution of the cycle is always accepted (this is also the case for the HyperGA technique). The choice function hyperheuristic presented in [Sou03] uses a quality control mechanic which restores a solution to a previously saved point if no absolute improvement is generated within a specific number of heuristic calls; in [Che07] a similar

mechanic knowingly alters the weights of hard constraint violations in its solutions in order to facilitate diversification of search.

Finally, we demonstrate that even simple hyperheuristics may be competitive with real-world problems, and that there remains a need to analyse the strengths of different hyperheuristics as applied to different problems, identifying salient features which support the use of one black-box hyperheuristic over another.

References

[AK03] M. Ayob and G. Kendall. A monte-carlo hyperheuristic to optimise component placement sequencing for multi head placement machine. In *Proceedings of the International Conference on Intelligent Technologies (InTech'03)*, pages 132–141, Chiang Mai, Thailand, December 17-19 2003.

[AMHV03] A. Anagnostopoulos, L. Michel, P. Van Hentenryck, and Y. Vergados. A simulated annealing approach to the traveling tournament problem. In *Proceedings of CPAIOR'03*, 2003.

[Aro96] L. D. Aronson. Algorithms for vehicle routing - a survey. Technical report, Delft, The Netherlands, 1996.

[Bai05] R. Bai. *An Investigation of Novel Approaches for Optimising Retail Shelf Space Allocation*. PhD thesis, School of Computer Science, University of Nottingham, 2005.

[BBG+07] R. Bai, E. K. Burke, M. Gendreau, G. Kendall, and B. McCollum. Memory length in hyper-heuristics: An empirical study, a combined constructive improvement heuristic for examination timetabling. In *Proceedings*

of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CISched2007), Hilton Hawaiian Village, Honolulu, Hawaii, USA, 1-5 April 2007.

[BDPQ05] E. Burke, M. Dror, S. Petrovic, and R. Qu. Hybrid graph heuristics in a hyper-heuristic approach to exam timetabling problems. In *The Next Wave in Computing, Optimization and Decision Technologies (eds. B.L. Golden, S. Raghavan and E.A. Wasil)*, pages 79–92. Springer, 2005.

[BF03] A. Badr and A. Fahmy. A proof of convergence for ant algorithms. *Information Sciences*, 2003.

[BHK+03] 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, pages 457–474. Kluwer Academic Publishers, 2003.

[BHK06] E. K. Burke, M. R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN IX)*, LNCS 4193, Reykjavik, Iceland, September 9-13 2006. Springer.

[BHKW07a] E. K. Burke, M. R. Hyde, G. Kendall, and J. R. Woodward. Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In H. Lipson and D. Thierens, editors, *Proceedings*

of the 2007 Genetic and Evolutionary Computation Conference (GECCO '07), London, UK, July 7-11 2007. ACM 2007.

[BHKW07b] E. K. Burke, M. R. Hyde, G. Kendall, and J. R. Woodward. The scalability of evolved online bin packing heuristics. In *Proceedings of the Congress on Evolutionary Computation (CEC 2007)*, Swissotel The Stamford, Singapore, September 25-28 2007.

[BJKW97] E. Burke, K. Jackson, J.H. Kingston, and R. Weare. Automated university timetabling: The state of the art. *The Computer Journal*, 40(9):565–571, 1997.

[BK03] R. Bai and G. Kendall. An investigation of automated planograms using a simulated annealing based hyper-heuristics. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Proceedings of The Fifth Metaheuristics International Conference (MIC 2003)*, Kyoto International Conference Hall, Kyoto, Japan, 23-25 August 2003. Springer.

[BK05] E. Burke and G. Kendall, editors. *Search Methodologies: Introductory Tutorials in Optimisation and Decision Support Methodologies*. Springer, 2005.

[BKO+03] E. Burke, G. Kendall, R. O'Brien, D. Redrup, and E. Soubeiga. An ant algorithm hyper-heuristic. In *Proceedings of the Fifth Metaheuristics International Conference 2003 (MIC 2003)*, Kyoto, Japan, 25-28 August 2003.

[BKS03] E. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[BKS+05] E. Burke, G. Kendall, J.D. Landa Silva, R. O'Brien, and E. Soubeiga. An ant algorithm hyperheuristic for the project presentation scheduling problem. In *Proceedings of the Congress for Evolutionary Computation 2005 (CEC2005)*, volume 3, pages 2263–2270, 2005.

[BOK06] B. Bilgin, E. Ozcan, and E. E. Korkmaz. An experimental study on hyper-heuristics and exam timetabling. In *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pages 123–140, Brono, Czech Republic, August 30-September 1 2006.

[Bow03] K. Bowen. Sixty years of operational research. *European Journal of Operational Research*, 153(3):618–623, 16 March 2003.

[BPQ06] E. Burke, S. Petrovic, and R. Qu. Case based heuristic selection for timetabling problems. *Journal of Scheduling*, 9:115–132, 2006.

[CC03] P. Cowling and K. Chakhlevitch. Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC'2003)*, pages 1214–1221, Canberra, Australia, 2003. IEEE Computer Society Press.

[CC05] K. Chakhlevitch and P. Cowling. Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In *Proceedings of the Fifth European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP2005)*, Lausanne, Switzerland, 2005.

[CDM+96] A. Colorni, M. Dorigo, F. Maffioll, V. Maniezzo, G. Righini, and M. Trubian. Heuristics from nature for hard combinatorial optimisation problems. *International Transactions in Operational Research*, 3:1–21, 1996.

[Che07] P-C. Chen. *An Investigation of a Hyper Heuristic Ant Algorithm for the Travelling Tournament Problem, MRes Thesis*. School of Computer Science, University of Nottingham, 2007.

[CKB07] P-C. Chen, G. Kendall, and G. Vanden Berghe. An ant based hyper-heuristic for the travelling tournament problem. In G. Kendall, E. K. Burke, S. Smith, and K. C. Tan, editors, *Proceedings of IEEE Symposium of Computational Intelligence in Scheduling (CI-Sched 2007)*, pages 19–26, Hawaii, 2007. IEEE Press.

[CKH02] P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the Congress on Evolutionary Computation 2002, CEC 2002*, pages 1185–1190, 2002.

[CKS01a] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Selected papers of Proceedings of the 3rd International Conference on the Practice And Theory of Automated Timetabling*, LNCS 2079, pages 176–190. Springer, 2001.

[CKS01b] P. Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyperheuristic

for scheduling a sales summit. In *Proceedings of the Fourth Metaheuristics International Conference (MIC 2001)*, pages 127–131, 2001.

[CKS02] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Proceedings of the 2nd European Conference on EVOlutionary computation for Combinatorial OPtimisation, EvoCop 2002*, LNCS 2279, pages 1–10. Springer, 2002.

[CO98] F. Comellas and J. Ozon. An ant algorithm for the graph colouring problem. In *ANTS'98 - From Ant Colonies to Artificial Ants; in proceedings of the First International Workshop on Ant Colony Optimization*, Brussels, 1998.

[DMC96] 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*, 26(1):1–13, 1996.

[Dow98] K. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Resarch*, 106:393–407, 1998.

[DSB07] K. Dowsland, E. Soubeiga, and E. Burke. A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of Operational Research*, 179:759–774, 2007.

[EJKS01] A.T. Ernst, H. Jiand, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: a review of applications, methods and models. *European Journal of Operational Research*, 2001.

[ENT01] K. Easton, G. Nemhauser, and M. Trick. The travelling tournament problem description and benchmarks. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP 2001)*, LNCS 2239, pages 580–584. Springer, 2001.

[ENT02] K. Easton, G. Nemhauser, and M. Trick. Solving the travelling tournament problem: A combined integer programming and constraint programming approach. In E. Burke and P. De Causmaecker, editors, *Practice And Theory of Automated Timetabling (PATAT 2002)*, LNCS 2740, pages 100–112. Springer, 2002.

[FRC93] H-L. Fang, P. Ross, and F. Corne. A promising genetic algorithm approach to job shop scheduling, rescheduling and open-shop scheduling problems. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 375–382, San Mateo, 1993. Morgan Kaufmann.

[FRC94] H-L. Fang, P. Ross, and F. Corne. A promising hybrid ga/heuristic approach for open-shop scheduling problems. In A. Cohn, editor, *Proceedings of ECAI 94: 11th European Conference on Artificial Intelligence*, pages 590–594. John Wiley and Sons Ltd, 1994.

[FT61] H. Fisher and G. L. Thompson. Probabilitistic learning combinations of local job-shop scheduling rules. In *Factory Scheduling Conference*, Carnegie Institute of Technology, May 10-12 1961.

[GL97] F. Glover and M. Laguna, editors. *Tabu search*. Kluwer Academic Publishers, 1997.

[GRK04] A. Gaw, P. Rattadilok, and R. Kwan. Distributed choice function hyper-heuristics for timetabling and scheduling. In *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT'04)*, pages 495–497, 2004.

[GS05] L. Di Gaspero and A. Schaerf. A tabu search approach to the traveling tournament problem. In *Proceedings of the 6th Metaheuristics International Conference (MIC-2005)*, pages 23–27, Vienna, Austria, 2005.

[Han05] L. Han. *An Investigation of a Genetic Algorithm Based Hyper-heuristic applied to Scheduling Problems*. PhD thesis, School of Computer Science, University of Nottingham, 2005.

[HK02] L. Han and G. Kendall. An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem. In *SEAL2002*, pages 267–271, 2002.

[HK03a] L. Han and G. Kendall. An investigation of a tabu assisted hyper-heuristic genetic algorithm. In *Proceedings of the Congress of Evolutionary Computation (CEC 2003)*, 2003.

[HK03b] L. Han and G. Kendall. Guided operators for a hyper-heuristic genetic algorithm. In T. D. Gedeon and L. C. C. Fung, editors, *Proceedings of AI-2003: Advances in Artificial Intelligence. The 16th Australian Conference*

on Artificial Intelligence (AI'03), pages 807–820, Perth, Australia, 3-5 Dec 2003.

[HM01] P. Hansen and N. Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[Hol75] J.H. Holland, editor. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

[Hoo96] J. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics 33-42*, 1996.

[HR98] E. Hart and P. Ross. A heuristic combination method for solving job-shop scheduling problems. In A.E. Eiben, T. Back, M. Schoenauer, and H.P. Schwefel, editors, *Parallel Problem Solving from Nature V*, LNCS 1498, pages 845–854. Springer, 1998.

[HRN98] E. Hart, P. Ross, and J. A. D. Nelson. Solving a real-world problem using a heuristically driven evolving schedule builder. In *Evolutionary Computing*, volume 6, pages 61–81, 1998.

[Ken07] G. Kendall. Scheduling english football fixtures over holiday periods. *Journal of the Operational Research Society*, pages 1–13, 2007.

[KGV83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing, 1983.

[KH04a] G. Kendall and N. Mohd Hussin. Tabu search hyper-heuristic approach to the examination timetabling problem at university technology mara. In E. Burke and M. Trick, editors, *Proceedings of the 5th international conference on the Practice and Theory of Automated Timetabling (PATAT)*, pages 199–217, Pittsburgg, USA, 18-20 August 2004.

[KH04b] G. Kendall and N. Mohd Hussin. An investigation of a tabu search based hyper-heuristic for examination timetabling. In G. Kendall, E. Burke, and S. Petrovic, editors, *Selected papers from MISTA 2003*. Kluwer Publication, 2004.

[KSC02] G. Kendall, E. Soubeiga, and P. Cowling. Choice function and random hyperheuristics. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL 2002)*, pages 667–671, Singapore, November 2002.

[LKW97] D. B. Leake, A. Kinley, and D. C. Wilson. A case study of case-based CBR. In *ICCBR*, pages 371–382, 1997.

[LLP01] N. Lagoudakis, M. Littman, and R. Parr. Selecting the right algorithm. In *Proceedings of the 2001 Fall Symposium Series: Using Uncertainty within Computation*, Cape Cod, MA, USA, November 2001.

[McC06] B. McCollum. A perspective on bridging the gap between research and practice in university timetabling. In *Proceedings of the 6th International*

Conference on the Practice and Theory of Automated Timetabling, PATAT 2006, Brno, August 2006. LNCS 3867.

[Nar03] A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristics: Computer Decision-Making*, pages 523–544. Kluwer Academic Publishers, 2003.

[OBK06] E. Ozcan, B. Bilgin, and E. E. Korkmaz. Hill climbers and mutational heuristics in hyperheuristics. In R. Poli, C. Cotta, C. A. C. Coello, M. Pelikan, H. Ishibuchi, K. Sastry, D. Whitley, E. Zitzler, M. Sebag, and B. Sendhoff, editors, *Parallel Problem Solving from Nature - PPSN IX: Selected Papers from the 9th International Conference*, LNCS 4193, pages 202–211, Berlin, 2006. Springer.

[PQ02] 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)*, volume 82, pages 336–340, Milan, Italy, September 16-18 2002. IOS Press.

[QB08] R. Qu and E. K. Burke. Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society*, 2008.

[RDP03] O. Rossi-Doria and B. Paechter. An hyperheuristic approach to course

timetabling problem using an evolutionary algorithm. Technical report, Napier University, Edinburgh, Scotland, 2003.

[Ros05] P. Ross. Hyper-heuristics. In E. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimisation and Decision Support Methodologies*, chapter 17, pages 529–557. Springer, 2005.

[RSMBH02] P. Ross, S. Schulenburg, J. G. Marin-Blazquez, 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)*, pages 942–948, 2002.

[RSMBH03] P. Ross, S. Schulenburg, J. G. Marin-Blazquez, 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)*, pages 1295–1306, 2003.

[SG06] A. Schaerf and L. Di Gaspero. Measurability and reproducibility in timetabling research: State-of-the-art and discussion. In *Practice And Theory of Automated Timetabling VI (PATAT 2006)*, pages 53–62, 2006.

[Sou03] E. Soubeiga. *Development and Application of Hyperheuristics to Personnel Scheduling*. PhD thesis, School of Computer Science, University of Nottingham, 2003.

[TBGS98] S. Talukdar, L. Baerentzen, A. Gove, and P. De Souza. Asynchronous

teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4(4):295–321, 1998.

[TMFAR05] H. Terashima-Marin, E. J. Flores-Alvarez, and P. Ross. Hyper-heuristics and classifier systems for solving 2d-regular cutting stock problems. In *Proceedings of the Genetic and Evolutionary Computation Conference 2005*, pages 637–643, 2005.

[TMFZRVR06] H. Terashima-Marin, C. J. Farias-Zarate, P. Ross, and M. Valenzuela-Rendon. A ga-based method to produce generalized hyper-heuristics for the 2d-regular cutting stock problem. In *Proceedings of the Genetic and Evolutionary Computation Conference 2006*, pages 591–598, 2006.

[TMFZRVR07] H. Terashima-Marin, C. J. Farias-Zarate, P. Ross, and M. Valenzuela-Rendon. Comparing two models to generate hyper-heuristics for the 2d-regular bin packing problem. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO 2007)*, pages 2182–2189, 2007.

[Tri07] M. Trick. http://mat.gsia.cmu.edu/tourn/, 2007.

[WM97] D. Wolpert and W. G. MacReady. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

[WR95] A. Wren and J.-M. Rousseau. Bus driver scheduling - an overview. In J.R. Daduna, I. Branco, and J.M.P. Paixao, editors, *Computer Aided Transirt Scheduling*, pages 173–187. Springer-Verlag, 1995.

[Wre95] A. Wren. Scheduling, timetabling and rostering - a special relationship? In *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)*, pages 475–495. Napier University, 1995.