Volume 31 (1), pp. 39–60
http://orion.journals.ac.za

ORiON
ISSN 0529-191-X

A genetic algorithm selection perturbative hyper-heuristic for solving the school timetabling problem

R Raghavjee* N Pillay†

Received: 29 August 2013; Revised: 14 August 2014; Accepted: 15 August 2014

Abstract

Research in the domain of school timetabling has essentially focused on applying various techniques such as integer programming, constraint satisfaction, simulated annealing, tabu search and genetic algorithms to calculate a solution to the problem. Optimization techniques like simulated annealing, tabu search and genetic algorithms generally explore a solution space. Hyper-heuristics, on the other hand, search a heuristic space with the aim of providing a more generalized solution to the particular optimisation problem. This is a fairly new technique that has proven to be successful in solving various combinatorial optimisation problems. There has not been much research into the use of hyper-heuristics to solve the school timetabling problem. This study investigates the use of a genetic algorithm selection perturbative hyper-heuristic for solving the school timetabling problem. A two-phased approach is taken, with the first phase focusing on hard constraints, and the second on soft constraints. The genetic algorithm uses tournament selection to choose parents, to which the mutation and crossover operators are applied. The genetic algorithm selection perturbative hyper-heuristic (GASPHH) was applied to five different school timetabling problems. The performance of the hyper-heuristic was compared to that of other methods applied to these problems, including a genetic algorithm that was applied directly to the solution space. GASPHH performed well over all five different types of school timetabling problems.

Key words: Timetabling, hyper-heuristics, combinatorial optimisation, evolutionary algorithms.

1 Introduction

Educational timetabling encompasses university examination, university course and school timetabling. These problems vary in terms of requirements and constraints to such an extent that methods providing a solution to one type of timetabling problem may not be

*School of Management, IT and Governance, University of KwaZulu-Natal, Pietermartizburg Campus, Pietermaritzburg, 3201

†Corresponding author: School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal, Pietermaritzburg Campus, Pietermaritzburg, 3201, email: pillayn32@ukzn.ac.za

http://dx.doi.org/10.5784/31-1-158

effective in solving the other two types. Thus, these three timetabling problems have been studied as different optimisation problems [19, 29, 30]. The school timetabling problem has been fairly well researched. Various optimisation methods have been applied to this problem including bee algorithms, constraint satisfaction methods, cyclic transfers, evolutionary algorithms, GRASP, integer programming, neural networks, simulated annealing and tiling algorithms [27]. School timetabling problems differ drastically as they are dependent on the educational system of the particular country [28]. Thus, a method that performs well in solving one school timetabling problem may not produce good results for another. Hyper-heuristics, defined as "heuristics to choose heuristics", aim at providing a generalized solution[1] to a particular problem, instead of the best solution for one or more problem instances [11].

Hyper-heuristics have proven to be effective in solving combinatorial optimisation problems. Selection perturbative hyper-heuristics have produced good results for the maximum satisfiability, one-dimensional bin-packing, permutation flow shop, personnel scheduling, travelling salesman, and vehicle routing problems, and have been able to provide a generalized solution across all six of these domains [21]. Previous research into solving the school timetabling problem using genetic algorithms revealed that the most effective mutation operators were problem dependent, *i.e.* different operators produced the best results for each school timetabling problem. Thus, it is anticipated that a selection perturbative hyper-heuristic would be effective in providing a solution to the school timetabling problem that is capable of generalizing over a set of problems, with different sets of low-level heuristics used for the different types of school timetabling problems.

There has not been much research into the use of hyper-heuristics for solving the school timetabling problem. Given the good performance of selection perturbative hyper-heuristics in solving other combinatorial optimisation problems, this paper investigates the use of a genetic algorithm selection perturbative hyper-heuristic (GASPHH) for the school timetabling problem. GASPHH was evaluated on five different school timetabling problems, and was able to produce feasible solutions of good quality for each problem. The results produced by GASPHH were found to be better than other methods applied to the same problem instances.

The remainder of this paper is structured as follows. The following section provides an overview of the background to this study in terms of school timetabling, evolutionary algorithms, hyper-heuristics, and previous work applying hyper-heuristics to the school timetabling problem. In §3 the logic of GASPHH is described. The experimental setup used to test GASPHH is presented in §4. In §5 the performance of GASPHH to solve school timetabling problems is discussed. The findings of this study and future extensions of this work are summarized in §6.

[1] Please note that in the context of this study a general solution refers to a method providing a solution to a particular problem domain by producing solutions specific to each problem instance. It does not mean that a single solution produced by the method is a solution for a number of problem instances.

2 Background

This section provides an overview of the school timetabling problem, hyper-heuristics, previous work applying hyper-heuristics to the school timetabling problem, and evolutionary algorithm hyper-heuristics.

2.1 The school timetabling problem

The school timetabling problem essentially requires the allocation of resources to timetable periods, so as to satisfy the problem hard constraints, and minimise the soft constraint violations [27]. The resources include teachers, venues and classes. In some problems, venues are not taken into consideration. In order for a timetable to be feasible it must meet all the hard constraints. The soft constraint cost is usually minimised, and is a measure of the quality of the timetable.

The hard and soft constraints differ vastly from one school timetabling problem to the next, and what may be defined as a hard constraint for one problem could be regarded as a soft constraint for another, and *vice versa*. Pillay [27] classifies constraints into seven categories, namely, problem requirements constraints, no clashes constraints, resource utilization constraints, workload constraints, period distribution constraints, preference constraints and lesson constraints. Problem requirement constraints are hard constraints that are defined by the requirements of the problem and have to be satisfied, *i.e.* a particular teacher has to meet with a particular class for a specified number of times per week to present a lesson in a specific subject. The second category, no clashes constraints, is also a hard constraint which require no resources to be scheduled more than once in a particular period. Resource utilization constraints are hard constraints which ensure that resources are not under or over utilized, *e.g.* the capacity of a venue should not be exceeded, teachers should not be scheduled when unavailable. Workload constraints are problem dependent, and are defined as hard constraints for some problems and soft constraints for others. Examples of this category of constraints include teacher workloads must not be exceeded, a maximum and minimum limit on the number of lessons a class should have per day. Period distribution constraints restrict the distribution of periods for classes and/or teachers, *e.g.* classes of a certain grade should not have idle or free periods or these should be scheduled in the last two periods of the school day, only one lesson on a particular subject should be scheduled per day per class. These constraints have been found to be hard or soft depending on the problem instance. Preference constraints are soft constraints indicating teacher or lesson preferences, *e.g.* Mathematics should be scheduled early in the school day, teachers may have a preference not to teach in certain periods. The last category of constraints, lesson constraints, are hard constraints that pertain to certain requirements that must be met for a specific lesson, *e.g.* certain lessons should take place simultaneously or in succession, splitting and merging of classes for certain lessons.

There has been much research into solving various school timetabling problems. This has been specific to a particular school timetabling problem, *e.g.* the Greek or German school timetabling problem which has been defined based on the educational system of the particular country. Solving the school timetabling problem involves assigning class-teacher or class-teacher-venue tuples to timetable periods. Each tuple represents a lesson given

to a class by a teacher on a particular subject. If a teacher has to meet a class for a lesson on a subject x times a week, x tuples have to be allocated to the timetable. Tuples are chosen for allocation either randomly or according to the difficulty of allocating the tuple. A low-level construction heuristic is used to calculate the difficulty of scheduling a tuple. The two construction heuristics generally used for the school timetabling problem are the number of lessons per tuple and the number of available timetable periods the tuple can be allocated to [27]. These heuristics are synonymous to the graph colouring heuristics largest degree and saturation degree respectively [30]. Methods that have been applied in the various studies to allocate tuples or improve the feasibility and/or quality of a timetable created using one of the low-level heuristics include bee algorithms, constraint programming, constraint satisfaction methods, cyclic transfers, evolutionary algorithms, GRASP, integer programming, neural networks, simulated annealing, tabu search, threshold acceptance methods, tiling algorithms, the walk up jump down algorithm, as well as various hybrid approaches [27], and more recently particle swarm optimisation [36, 37]. However, in a majority of these studies a particular method has been applied to a specific problem without any focus on generalization and evaluation on more than one school timetabling problem type.

2.2 Hyper-heuristics

Hyper-heuristics are "heuristics" to select or generate low-level heuristics and aim to provide a more generalized solution for a problem domain. The low-level heuristics can be constructive or perturbative. Construction heuristics are used to construct a solution to a combinatorial optimisation problem, while perturbative heuristics are used to improve an initial potential solution which is usually created using a construction heuristic. The two types of low-level heuristics are best illustrated with an example from the domain of examination timetabling. Construction heuristics generally used to create an examination timetable are graph colouring heuristics, namely, largest degree, largest weighted degree, largest colour degree, largest enrolment and saturation degree [30]. These heuristics are an estimate of the difficulty of scheduling an examination. Examinations to be scheduled are usually sorted according to one of these heuristics and allocated in order to create a timetable. Perturbative low-level heuristics are moves or operators that will be applied to the initially created timetable to improve feasibility and/or quality and include swapping of exams, swapping of rows or columns in a timetable, de-allocating exams and allocating unscheduled examinations.

Burke *et al.* [12] define four categories of hyper-heuristics, *i.e.* selection constructive, selection perturbative, generation constructive and generation perturbative. Selection hyper-heuristics choose the construction heuristic to apply during each step of the construction of a solution, or which perturbative heuristic to apply next during each stage of improvement. Methods used by selection constructive hyper-heuristics to choose construction heuristics include hill-climbing, genetic algorithms, tabu search, iterated local search, variable neighbourhood search, fuzzy logic, case-based reasoning, classifier systems and messy genetic algorithms [10]. Burke *et al.* [10] describe selection perturbative hyper-heuristics as being single point search based or population based. The former attempts to improve a single candidate solution on each iteration. This is achieved by means of two processes, namely

heuristic selection and move acceptance. Various methods have been used for both these processes. Burke *et al.* [10] categorize heuristic selection methods into those with learning and those without. Heuristic selection without learning essentially refers to variations of randomly selecting the low-level perturbative heuristic to apply next, while methods with learning include the choice function and variations of reinforcement learning. Techniques generally used for move acceptance are defined as deterministic or non-deterministic. Deterministic move acceptance methods either accept all moves or select moves resulting in equal and/or improved cost; while methods for non-deterministic acceptance include Monte Carlo, great deluge, tabu search, simulated annealing and late acceptance [10]. Population based selection pertubative heuristics do not use separate techniques for heuristic selection and move acceptance and both are achieved by the population based method. Population based techniques employed by selection perturbative hyper-heuristics are genetic algorithms and ant colony optimisation. Generation hyper-heuristics, on the other hand, generate a low-level heuristic using existing heuristics and/or problem characteristics. Genetic programming has chiefly been used for this purpose [10]. Essentially genetic programming produces a new low-level heuristic which is constructive (generation constructive hyper-heuristics) or perturbative (generation perturbative hyper-heuristics). The generated low-level heuristic may be disposable or reusable. Disposable heuristics are created for a particular problem instance, while reusable heuristics are created offline and can be applied to a set of problem instances; however, there is no guarantee that the heuristic will perform well on new instances of the problem.

The availability of the HyFlex architecture [9] has contributed to the advancement of selection perturbative hyper-heuristics. HyFlex works at a higher level of abstraction and is domain independent. HyFlex should be able to solve a new instance of a problem given a different set of low-level perturbative heuristics and evaluation function. Low-level perturbative heuristics for 6 combinatorial optimisation domains, namely, maximum satisfiability, one dimensional bin-packing, permutation flow shop, personnel scheduling, travelling salesman and vehicle routing are available for use with HyFlex. This enables researchers using HyFlex to concentrate on developing domain independent methods for heuristic selection and move acceptance. The low-level heuristics available in HyFlex are divided into four groups, namely mutational, ruin and create, hill-climbing/local search and crossover heuristics. HyFlex allows for generalization within the specific domain and across all 6 domains [21].

Given the progress made with selection perturbative hyper-heuristics and their effectiveness in solving combinatorial optimisation problems, this paper will investigate the use of a selection perturbative hyper-heuristic in solving the school timetabling problem. A population based method, namely a genetic algorithm, will be used to explore the space of low-level perturbation heuristics. This will allow for different sets of low-level heuristics to be used for the different school timetabling problems. The following section provides an overview of previous work applying hyper-heuristics to the school timetabling problem.

2.3 Hyper-heuristics and the school timetabling problem

Scholarly research on the use of hyper-heuristics to solve the school timetabling problem appears to be in its infancy. Pillay [22, 23] has employed a selection constructive hyper-

heuristic to solve the school timetabling problem. The hyper-heuristic was used to induce a solution to one of the problems in the Abramson benchmark set [35] and that of a South African primary school. The hyper-heuristic employs an evolutionary algorithm to explore a space of combinations of low-level construction heuristics, namely largest degree, saturation degree, double degree and period preference degree. The hyper-heuristic produced feasible timetables for both problems.

Pillay [24] examines the use of a generation constructive hyper-heuristic to solve the school timetabling problem. Genetic programming is used to generate low-level construction heuristics for the school timetabling problems. Each evolved heuristic is a program combining existing low-level construction heuristics, arithmetic operators and conditional statements. One or more of the following low-level heuristics are included in each program: largest degree, saturation degree, class degree and teacher degree. The evolved low-level heuristics are disposable. The hyper-heuristic was tested on two problems from the Abramson benchmark set. The evolved heuristic was found to perform better than the largest degree and saturation degree heuristics in solving these problems.

2.4 Evolutionary algorithms and hyper-heuristics

From the previous sections it is evident that the two evolutionary algorithms that have played a significant role in hyper-heuristics are genetic algorithms [16] and genetic programming [18]. Both GAs and GP first create an initial population of randomly created chromosomes or individuals, which are then iteratively refined via the processes of fitness evaluation, selection and regeneration. A fitness function is used to calculate the fitness of each individual and is application dependent. Selection methods commonly used are fitness proportionate selection and tournament selection, which are used to choose parents of successive generations. Crossover, mutation and reproduction are usually used to create offspring. Genetic algorithms explore a solution space, while genetic programming searches a space of programs. Genetic algorithms evolve a solution to a problem while genetic programming evolves a program, which when implemented produces a solution. Thus, each individual in a GP population is a parse tree representing a program.

Genetic programming is chiefly employed by generation hyper-heuristics [10]. Each program in the population represents the algorithm for a constructive or perturbative heuristic. The program is evaluated by using the heuristic to solve the problem in question, and generally the fitness is a measure of how far the solution produced is from the known optimum.

Genetic algorithms have been used in selection hyper-heuristics [17, 25, 26, 34] to explore a space of heuristic combinations. Each chromosome is a string composed of numeric or alphabetic characters, with each character representing a low-level heuristic. As in the case of generation hyper-heuristics, the fitness of each chromosome is determined by applying the chromosome to either creating a solution in the case of selection constructive hyper-heuristics, or improving a candidate solution for selection perturbative hyper-heuristics. The cost of the solution or candidate solution is used as a fitness measure. The cost is specific to the problem domain. For example, in the context of school timetabling this would be the number of hard and/or soft constraint violations in the constructed or improved timetable. Genetic algorithm selection perturbative hyper-heuristics can also be viewed as

a genetic algorithm using an indirect representation as opposed to a direct representation whereby the GA explores a space of candidate solutions, and genetic operators are applied directly to these [1].

The following section describes the genetic algorithm selection perturbative hyper-heuristic implemented to solve the school timetabling problem.

3 Genetic algorithm selection perturbative hyper-heuristic

From literature on the school timetabling problem [5, 6, 15, 20], it is evident that a two-phased approach, with the first phase focusing on hard constraints and the second phase on soft constraints, is effective in solving this problem. Thus, this study takes the same approach with both phases employing the same genetic algorithm selection perturbative hyper-heuristic, with the low-level heuristics differing for each phase. The GA hyper-heuristic for both phases explores a space of heuristic combinations represented by strings. A generational genetic algorithm, illustrated in Algorithm 1, is employed and iteratively refines an initial population of randomly created strings via the processes of evaluation, selection and regeneration. These processes are described in the sections that follow. Each iteration is referred to as a generation, and g generations denote a run. The termination criterion for Phase 1 is that a feasible solution is found. For Phase 2 a preset number of maximum generations is used as a termination criterion.

Algorithm 1: Generational genetic algorithm implemented.

```
1 Create an initial population
2 repeat
3 |   Evaluate the population assigning a fitness measure to each individual
4 |   Select parents
5 |   Apply the genetic operators to selected parents to produce offspring
6 until the termination criterion is met
```

3.1 Initial population generation

Each chromosome is a string comprised of low-level heuristics. The length of the string is equal to the total number of class-teacher meetings to be scheduled. The chromosomes in the initial population have a fixed length but are variable length in successive generations as a result of the application of the crossover operator. Each chromosome is created by randomly choosing a low-level heuristic for each position in the chromosome. The perturbative low-level heuristics used, and initial population generation for each of the two phases are described below.

3.1.1 Low-level perturbative heuristics

Two factors have played a role in determining the set of low-level perturbative heuristics to use, namely the categories of low-level heuristics defined for the HyFlex architecture, and secondly the operators used in previous studies to create or improve school timetables. The HyFlex architecture includes ruin and create operators. Allocate and deallocate

operators are included to cater for this. The allocate operator schedules a tuple to a feasible timetable period. If there is more than one feasible period available, the tuple is allocated to the period that will result in the lowest soft constraint cost. If a feasible period cannot be found, the tuple is allocated to a randomly chosen period. If there are no tuples to allocate this heuristic has no effect. The deallocate operator removes a randomly chosen tuple from the timetable, and adds it to the list of the tuples to be scheduled. If the timetable is empty, the deallocate heuristic has no effect. As in the HyFlex architecture mutational heuristics are also included. From previous research on solving the school timetabling problem, operations performed to improve a timetable essentially swap the tuples between periods [8, 15, 20]. Tuples are either randomly chosen, or alternatively one or both tuples that cause constraint violations are chosen. The mutation low-level heuristics used in this paper are based on this methodology. In this paper, each timetable is a matrix with the rows corresponding to periods and the columns to classes. Each cell contains either a teacher or a teacher-venue tuple if the problem involves venue allocation. The following mutational heuristics are made available:

- Two violation mutation - Two cells in the matrix causing constraint violations are identified, and the contents of the cells are swapped.
- One violation mutation - The contents of a cell resulting in a constraint violation is swapped with that of a randomly selected cell.
- Random swap - Two cells are randomly selected and the contents swapped.
- Row swap - The contents of the cells in two randomly selected rows are swapped.
- One violation row swap - The contents of a row containing a tuple causing a violation is swapped with that of a randomly chosen row.

Each of the mutational heuristics perform s swaps. Only cells in the same column are swapped, to prevent teachers from being scheduled to teach classes they are not allocated to teach.

The benefit of incorporating hill-climbing into operators for school timetabling is evident from previous research in this domain [3, 13, 15, 33].Versions of these mutational heuristics with hill-climbing are also made available. An approach similar to that taken by Cerderia-Pena [13] and Rahoual [33] is employed, with the hill-climbing implemented within the mutation process. While the timetable is being mutated, the currently mutated version is compared to either the parent or the timetable from the previous swap (if this swap has improved fitness). If the fitness of the new version is better than the current version, then the swap is accepted. If the older version has a better fitness then the swap is rejected. A limit is set on the number of attempts at producing a fitter offspring in order to prevent premature convergence and long computational times. A value of fifty was found to be sufficient. Both the mutational heuristics and the mutational heuristics incorporating hill-climbing have no effect if there have been no tuples allocated to the timetable at the time of application of the perturbative heuristic. The general algorithm for the mutational heuristics is shown in Algorithm 2. The Mutate Offspring instruction in lines 6 and 15 of Algorithm 2 will vary depending on the mutation heuristic being used.

From previous work on the school timetabling problem the use of crossover to improve timetables produced duplicate or missing tuples which made it necessary to incorporate

Algorithm 2: General algorithm for mutation heuristics

```
1  if Hill climbing is used then
2      HillClimbingSteps=0
3      while (ParentFitness < OffspringFitness) AND (HillClimbingSteps < 50) do
4          Offspring = Copy parent
5          for MutLoop = 1 to Swaps do
6              Mutate offspring
7              Calculate offspring fitness
8              if Offspring fitness < Previous offspring fitness then
9                  Accept swap
10             else
11                 Reject swap
12             HillClimbingSteps++
13 else
14     for MutLoop = 1 to Swaps do
15         Mutate Offspring
16         Evaluate Offspring
```

repair mechanisms resulting in an additional processing overhead [3, 8, 14, 39]. Thus, it was decided not to use crossover perturbative heuristics in this study.

3.1.2 Phase 1

Each element of the initial population is created by firstly creating n chromosomes, where each chromosome is created randomly. *ADA3DA4AA34DDDA* is an example of a chromosome. An *A* denotes the allocate heuristic, *D* deallocate, *3* the mutation heuristic that swaps the contents of a tuple involved in a constraint violation with that of a randomly chosen tuple, and *4* the heuristic that swaps the contents of two tuples causing a hard constraint violation. The fitness of each chromosome is determined by using the chromosome to create a timetable. Each timetable is represented as a matrix with the rows representing periods and the columns classes. Each cell stores the teacher teaching the class during the period, or the teacher and venue in which the lesson will be held. The tuples to be allocated are sorted according to difficulty of scheduling the tuple. The saturation degree, *i.e.* the number of timetable periods that the tuple can be allocated to which does not result in hard constraint violations, is used to assess difficulty. A lower value is indicative of a tuple that is more difficult to schedule, and the tuples are sorted in ascending order of saturation degree for allocation. Each tuple consists of a class and teacher or a class, teacher and venue. Each tuple is allocated to a feasible timetable period. If a feasible period cannot be found, the tuple is allocated to a randomly selected period. If there is more than one feasible period which the tuple can be scheduled in, the period resulting in the minimum soft constraint cost is chosen, *i.e.* the minimum penalty slot. The algorithm for applying the chromosome to create a timetable is illustrated in Algorithm 3. The chromosome is traversed from left to right and each low-level perturbative heuristic is applied in sequence. Each chromosome is applied once. The fitness of the chromosome is the sum of the hard constraints that are violated in the constructed timetable, including the number of unallocated tuples.

Algorithm 3: Algorithm used to create a timetable using a chromosome

```
for each character in heuristic string do
    if character = A then
        if all tuples are not allocated then
            Find tuple that is most difficult to allocate (lowest saturation degree)
            Allocate tuple to timetable
            Resort unallocated tuple list in order of difficulty
    else if character = D then
        if timetable is nonempty then
            Randomly select tuple from timetable
            Remove tuple and return to unallocated tuple list
    else if character = 1 or 2 or 3 or 4 or 5 or 6 or 7 then
        if the hard (or soft) constraint cost ≠ 0 then
            Apply associated mutation operator
```

The chromosome amongst these with the best fitness, *i.e.* lowest number of hard constraint violations, then forms part of the initial population. The number of soft constraint violations is used to break ties in cases where there is more than one chromosome with the lowest number of hard constraint violations. If the initial population size is m then this process is performed m times. It is anticipated that this method would improve the overall fitness of the initial population, and thus improve the possibility of finding feasible timetables. If n is 1 the process becomes equivalent to that of the standard initial population generation employed by genetic algorithms.

Phase 1 is terminated once a feasible solution is found, or a maximum number of generations has been reached. If a feasible timetable is not found, the run is aborted and deemed unsuccessful.

3.1.3 Phase 2

The feasible timetable found in Phase 1 is kept in working memory. A population of chromosomes is created as in Phase 1; however, the set of low-level heuristics are different and aimed at minimising the soft constraint cost. Furthermore, each chromosome is randomly created and the best of n chromosomes is not determined, *i.e.* n has a value of 1 for Phase 2. Each chromosome is applied to the feasible timetable in working memory to improve the soft constraint cost of the feasible timetable, without introducing hard constraint violations. The main aim of the second phase is to evolve a chromosome that results in the largest reduction in the soft constraint cost of the feasible timetable produced by the chromosome induced in Phase 1.

3.2 Evaluation and selection

Each chromosome is evaluated by using it to construct (Phase 1) or improve (Phase 2) a timetable. The fitness of each chromosome is the hard constraint cost of the timetable in

Phase 1 and the soft constraint cost in Phase 2. The hard constraint violations include unallocated tuples, *i.e.* the number of class-teacher meetings not scheduled for each subject. Thus, chromosomes that do not schedule all meetings will be penalized.

Tournament selection [18] is used to choose parents which the genetic operators are applied to. A tournament of t individuals is randomly selected from the population and the fittest chromosome is returned as a parent.

3.3 Regeneration

Mutation and crossover are used to create the offspring of successive generations. Mutation is applied by randomly choosing a character in the chromosome and replacing it with a randomly selected character from the set of low-level heuristics. Suppose the following chromosome is chosen as a parent and the underlined character the mutation point *341<u>2</u>12AD1A*. The *2* will be replaced with a randomly chosen low-level heuristic. For example, if this is an *A*, the corresponding offspring will be *341<u>A</u>12AD1A*.

The cut and splice crossover operator defined by Goldberg [16] is used and creates offspring of variable length. Different crossover points are selected in each parent. Each point is randomly selected. Both parents are crossed over at these points to create two offspring. Suppose that *ADADAAA341* and *21444ADADA* have been chosen as parents, with 4 the crossover point in the first parent and 8 in the second parent. The corresponding offspring are *ADADDA* and *21444ADAAAA341*.

The percentage of successive generations created using each of the genetic operators is determined by the application rates set at the beginning of the genetic algorithm run. The most appropriate application rates to use is problem dependent.

4 School timetabling problem list

GASPHH was tested on five different timetabling problems:

- The Abramson benchmark set
- Two different Greek high school timetabling problems
- A South African primary school timetabling problem
- A South African high school timetabling problem

Each of these problems is described in the following sections. The fitness function for all problems in Phase 1 is

$$f(t) = \sum_{x=1}^{N} H_x \geq 0, \tag{1}$$

where H_x is the number of violations by timetable t for hard constraint x, with N being the total number of hard constraints. The objective is to minimise $f(t)$ and a feasible timetable is found if $f(t) = 0$. The same fitness function is used for all problems. A chromosome producing a feasible timetable will have a fitness value of zero.

The formulation of the fitness function to calculate the soft constraint cost in Phase 2 is a summation of the number of violations for each of the listed soft constraints and is given by

$$h(t) = \sum_{y=1}^{M} S_y \geq 0, \tag{2}$$

where S_y represents the number of soft constraint violations in the timetable for soft constraint y, with M being the total number of soft constraints. A lower soft constraint cost indicates a better quality timetable. Therefore the objective is to minimise the soft constraint cost $h(t)$. This fitness function is used for all problems.

Note that it is not necessary to perform Phase 2 if soft constraints are not defined for the problem (*i.e.* all the constraints for the problem are hard constraints).

4.1 Abramson benchmark set

Abramson [2] has made available five artificial timetabling problems. These problems are hard timetabling problems (hence the acronym *hdtt*) as all periods must be utilized with very little or no options for each allocation. The characteristics of the problems are listed in Table 1. Each school week is comprised of five days, with six periods a day, giving a total of 30 timetable periods.

Problem	Number of teachers	Number of venues	Number of classes
hdtt4	4	4	4
hdtt5	5	5	5
hdtt6	6	6	6
hdtt7	7	7	7
hdtt8	8	8	8

Table 1: *A summary of the characteristics of the artificial school timetabling problems.*

All five problems have the following hard constraints:

- All class-teacher-venue tuples must be scheduled the required number of times (H_1^1).
- No class clashes, *i.e.* a class must not be scheduled more than once in a period (H_2^1).
- No teacher clashes, *i.e.* a teacher must not be scheduled more than once in a period (H_3^1).
- No venue clashes, *i.e.* a venue must not be allocated more than once to a timetable period (H_4^1).

4.2 The Greek school timetabling problem

GASPHH was also tested on the Greek school timetabling problems made available by Valouxis *et al.* [38] and Beligiannis *et al.* [7]. The problem presented by Valouxis *et al.* involves 15 teachers and 6 classes. There are 35 weekly timetable periods, *i.e.* 5 days with 7 periods per day. The hard constraints of the Valouxis *et al.* [38] problem set are:

- All class-teacher meetings must be scheduled (H_1^2).
- There must be no class or teacher clashes (H_2^2).
- Free/idle periods for classes must be scheduled in the last period of the day (H_3^2).
- Each teacher's workload limit for a day must not be exceeded (H_4^2).
- Class-teacher meetings must be uniformly distributed over the school week (H_5^2).

The problem soft constraints are:

- The number of free periods in the class timetable must be minimised (S_1^2).
- Teacher period preferences must be satisfied if possible (S_2^2).

Table 2 lists the characteristics of the Greek school timetabling problems made available by Beligiannis *et al.* [7]. There are 35 timetable periods per week. The sixth problem instance has not been used in studies subsequent to the study reported in [7] as it was found that not all data was available for this instance [36, 37, 40]. Thus, this instance has not been included in this study. Three of these problems have been included in the problem set of the third international timetabling competition focusing on high school timetabling [29].

Problem	Number of teachers	Number of classes	Number of co-teaching/subclass requirements
HS1	11	34	18
HS2	11	35	24
HS3	6	19	0
HS4	7	19	12
HS5	6	18	0
HS7	13	35	20

Table 2: *A summary of the characteristics of the Beligiannis problem set.*

The hard constraints for the Beligiannis *et al.* [7] problem set are:

- All class-teacher meetings must be scheduled (H_1^3).
- There must be no class or teacher clashes (H_2^3).
- Teachers must not be scheduled to teach when they are not available (H_3^3).
- Class free/idle periods must be scheduled in the last period of the day (H_4^3).
- Co-teaching and subclass requirements must be met (H_5^3).

The problem soft constraints are:

- The number of idle/free periods for teachers must be minimised (S_1^3).
- The uniform distribution of free periods for teachers (S_2^3).
- The workload for a teacher must be uniformly distributed over the week (S_3^3).
- Classes should not be taught the same subject in consecutive periods or more than once in a day if possible (S_4^3).

Note that while the study uses the same data set it includes an additional soft constraint listed in the first description of the problem in Beligiannis *et al.* [7] (p. 1267), namely the second constraint which is not used in previous studies.

4.3 South African primary school problem (SAPS)

This problem involves 19 teachers, 16 classes and 14 subjects. There are a maximum of 11 weekly timetable periods. However, different grades have a different number of daily periods ranging from 9 to 11. The hard constraints for the problem are:

- All required class-teacher meetings must be scheduled (H_1^4).
- There must be no class or teacher clashes (H_2^4).
- Certain subjects must be taught in specialized venues, *e.g.* Technology in the computer laboratory (H_3^4).
- Mathematics must be taught in the mornings – specified in terms of valid periods (H_4^4).
- All co-teaching requirements must be met (H_5^4).
- All double period requirements must be met (H_6^4).

The problem has one soft constraint, namely, the lessons per class must be uniformly distributed throughout the school week.

4.4 South African high school problem (SAHS)

The South African high school problem that the GA approach is applied to involves 30 classes, 40 teachers and 44 subjects. The hard constraints for the problem are:

- All required class-teacher meetings must be scheduled (H_1^5).
- There must be no class or teacher clashes (H_2^5).
- All sub-class and co-teaching requirements must be met (H_3^5).

The soft constraints for the problem are:

- Teacher period preferences must be met if possible (S_1^5).
- Period preferences for classes must be met if possible (S_2^5).

5 Experimental setup

Trials runs were conducted to determine the most appropriate values for the following genetic parameters:

- Population size – size of the population for each generation.
- Subpopulation size (n) – during Phase 1, each element of the population is created by first creating n chromosomes and the fittest chromosome forms part of the initial population.
- Number of generations
- Tournament size
- Number of swaps performed by mutational low-level perturbative heuristics
- Mutation application rate
- Crossover application rate

Problem	Sub-problem size	Population size	Tournament size	No. of swaps for mutational low-level heuristics	Generations	Mutation application rate	Crossover application rate
hdtt4–hdtt8	50	1000	10	200	50	20%	80%
Valouxis	50	1000	10	100	50	20%	80%
HS1–HS7	25	750	15	200	50	20%	80%
SAPS	20	500	10	200	50	20%	80%
SAHS	20	750	10	150	75	20%	80%

Table 3: *A summary of the genetic parameter values used for each data set to perform GASPHH.*

When testing each parameter value, thirty runs were performed. The most appropriate values found for each problem are listed in Table 3.

One of the aspects of hyper-heuristics that requires further research is the most appropriate set of low-level heuristics to use. A set containing too many low-level heuristics will increase the size of the heuristic space thereby increasing the time taken to reach an optimal heuristic combination, and a possible degradation in the performance of the hyper-heuristic. On the other hand, if the low-level heuristics that are needed to make an improvement are not included, this could result in the hyper-heuristic not being successful. In this paper, the set of low-level heuristics to use for each problem was empirically determined by conducting trial runs with elements of the three categories of low-level heuristics (ruin and create, mutational and hill-climbing) for each phase for each of the problems. Table 4 lists the sets that were found to be most effective for each of the different types of school timetabling problem. Please note the heuristics are defined as follows:

- A - Allocate a tuple from the list of tuples sorted according to saturation degree.
- D - Deallocate a tuple from timetable.
- 1 - Swap the contents of two cells involved in constraint violations. No hill-climbing.
- 2 - Swap the contents of two cells involved in constraint violations. No hill-climbing.
- 3 - Swap the contents of one cell involved in a constraint violation and a randomly chosen cell. Hill-climbing is incorporated in the mutation heuristic.
- 4 - Swap the contents of two cells involved in constraint violations. Hill-climbing is incorporated in the mutation heuristic.
- 5 - Swap the contents of two randomly chosen cells. Hill-climbing is incorporated in the mutation heuristic.
- 6 - Swap two randomly chosen rows. Hill-climbing is incorporated in the mutation heuristic.
- 7 - Swap the contents of a row containing a tuple causing a violation with that of a randomly chosen row. Hill-climbing is incorporated in the mutation heuristic.

GASPHH was developed using Visual C++ 2010. The random number generator function available in C++ is used to generate random numbers. A different seed is used for each run of GASPHH. Simulations (trial and final) were run in the Center for High Performance Computing[2].

[2]See http://www.chpc.ac.za/index.php/resources/tsessebe-cluster for cluster specifications.

Problem	Phase 1	Phase 2
HDTT4 - HDTT8	A, D, 3, 4	N/A
Valouxis	A, D, 3, 4	A, D, 3, 4, 5, 6
HS1 HS7	A, D, 1, 2, 3, 4	A, D, 3, 4, 5, 6
SAPS	A, D, 2, 3, 4	A, D, 3 4, 5
SAHS	A, D, 1, 2, 3, 4	A, D, 3, 5, 6, 7

Table 4: *The low-level heuristic sets for each problem.*

Problem	Success rate	Computational time of run producing best result	Average	Best soft constraint cost	Average soft constraint cost	Soft constraint standard deviation
HDTT4	100%	10 mins	11 mins	-	-	-
HDTT5	100%	26mins	20mins	-	-	-
HDTT6	100%	60 mins	68 mins	-	-	-
HDTT7	100%	240 mins	235 mins	-	-	-
HDTT8	100%	300 mins	678 mins	-	-	-
Valouxis	100%	120 mins	669 mins	34	34	0
HS1	100%	3 days	3 days	63	77.37	6.51
HS2	100%	600 mins	3 days	66	79	8.12
HS3	100%	3 days	3 days	17	18.47	0.86
HS4	100%	3 days	3 days	43	46.03	1.52
HS5	100%	90 mins	2 days	15	19.3	4.11
HS7	100%	1 day	3 days	98	105.4	4.95
SAPS	93%	1 day	7 days	3	7.61	2.81
SAHS	100%	3 mins	1.5 days	2	2.37	0.56

Table 5: *A summary of the computational performance of GASPHH. Experiments were performed in The Center for High Performance Computing.*

6 Results and discussion

This section discusses the performance of GASPHH on the five school timetabling problems listed in Section 4. Thirty runs were performed for each problem instance, with each run comprising both Phase 1 and Phase 2. In those cases where either the problem instance did not include soft constraints or a feasible timetable could not be found in Phase 1, only Phase 1 was performed. In the case of the latter, the run was deemed unsuccessful. Table 5 lists the success rate, runtimes, best soft constraint cost, average soft constraint cost, and soft constraint cost standard deviation, for each of the problem instances. As indicated in Table 5, GASPHH was able to find a feasible solution on all thirty runs for all problems except the Lewitt school timetabling problem. For this problem, feasible timetables were not produced for two runs, with the best timetable having one hard constraint violation for the one run and two violations for the other run. This can be attributed to random noise which is characteristic of stochastic methods such as genetic algorithms. Each heuristic combination was disposable and specific to each phase of a run as the reusable entity in this domain is the timetable created using the heuristic combination. As can be anticipated, the runtimes of GASPHH are higher than that of traditional methods generally applied to the school timetabling problem as a solution has to be created, and evaluated for each element of the population for each generation [25].

The performance of GASPHH was compared to other methods applied to the same problem for each of the five problems. In most cases, the comparison has been empirical as there

Method		HDTT4	HDTT5	HDTT6	HDTT7	HDTT8
SA1	BC	Unknown	0	0	2	2
	AC	Unknown	0.67	2.5	2.5	8.23
SA2	BC	**0**	0	0	0	0
	AC	**0**	0.3	0.8	1.2	1.9
TS	BC	0	0	3	4	13
	AC	0.2	2.2	5.6	10.9	17.2
GS	BC	5	11	19	26	29
	AC	8.5	16.2	22.2	30.9	35.4
NN-T2	BC	0	0	0	0	0
	AC	0.1	0.5	0.8	1.1	1.4
NN-T3	BC	0	0	0	0	0
	AC	0.5	0.5	0.7	1	1.2
SA3	BC	**0**	**0**	**0**	**0**	0
	AC	**0**	**0**	**0**	**0**	0.4
GA	BC	**0**	**0**	**0**	0	0
	AC	**0**	**0**	**0**	1.067	1.733
GASPHH	BC	**0**	**0**	**0**	**0**	**0**
	AC	**0**	**0**	**0**	**0**	**0**

Table 6: *A comparison for the Abramson data set, with BC the best number of hard constraint violations and AC the average number of hard constraint violations. The best results are highlited in bold.*

is insufficient data available for a formal comparison.

The performance of GASPHH was compared to the following methods applied to the Abramson benchmark set:

- SA1 A simulated annealing method implemented by Abramson *et al.* [4].
- SA2 A simulated annealing algorithm implemented by Randall [32].
- TS A tabu search employed by Randall [32].
- GS The greedy search method used by Randall [32].
- NN-T2 A neural network employed by Smith *et al.* [35].
- NN-T3 A second neural network employed by Smith *et al.* [35].
- SA3 - Simulated annealing with a sequence swap neighbourhood operator [40].
- GA - A genetic algorithm implemented by the authors to evolve school timetables [31].

The hard constraints for this set of problems are listed in Section 4. The minimum and average hard constraint costs for each of these methods and GASPHH are listed in Table 6. In both this study and the GA study [31] the average is taken over thirty runs. The best results are highlighted in bold.

GASPHH has performed well on all five problem instances, producing feasible solutions on all of the thirty runs. As more data was available for the GA study than the other methods, it was possible to conduct hypothesis tests to determine the statistical significance of the result that GASPHH performed better than the GA in solving the five problem instances. Table 7 lists the success rates for both methods. This result was found to be significant at the 1% level of significance.

Problem instance	Success rate	Success rate
HDTT4	100%	100%
HDTT5	100%	100%
HDTT6	100%	100%
HDTT7	100%	47%
HDTT8	100%	13%

Table 7: *A comparison of the GASPHH and GA algorithms' performance.*

Problem	GA	GASPHH
HS1	96	63
HS2	99	66
HS3	34	17
HS4	59	43
HS5	40	15
HS7	117	98

Table 8: *Performance comparison of the number of soft constraint violations for the Beligannis data set [7] with four soft constraints.*

Two studies attempted to solve the Greek school timetabling problem presented by Valouxis *et al.* [38], namely the constraint programming solution implemented by Valouxis *et al.* and the genetic algorithm presented in Raghavjee & Pillay [31]. Feasible timetables were produced by all three methods. The timetable produced by constraint programming had 45 soft constraint violations, the GA 35 soft constraint violations and GASPHH 34 violations. The result that GASPHH performs better than the GA was found to be significant at the 1% level of significance. Both GASPHH and the GA have obtained a 100% success rate, however the quality of the timetables produced by GASPHH is better than the GA, with GASPHH obtaining an average quality of 34 over the thirty runs and the GA 41.97.

The performance of GASPHH on the instances used by Beligainnis *et al.* [7] is compared to that of the genetic algorithm in [31]. Both methods produced feasible timetables for the 6 problems tested. The soft constraint costs of the best timetable produced by each method are listed in Table 8.

GASPHH has produced timetables of better quality in terms of constraint violations than the GA as reported in Table 8. Both GASPHH and the GA have produced feasible timetables for all thirty runs, but the quality of timetables produced by GASPHH is better than that of the GA. This was found to be significant at the 1% level of significance.

The implementation of this problem in this study differs from previous work in that it uses an additional soft constraint which previous studies have not taken into consideration, namely, uniform distribution of idle/gap periods for teachers. For completeness and to allow comparison with previous work, GASPHH was applied to the problem with the three constraints used in other studies and was able to find feasible solutions for all six problems. Table 9 lists the soft constraint cost of the best timetables produced by GASPHH and other methods applied to the same problem. The best soft constraint cost is highlighted in bold. The performance of GASPHH is compared to the evolutionary algorithm im-

Problem	EA[7]	PSO [36]	SA [40]	GASPHH
HS1	70	**15**	29	60
HS2	76	**17**	37	66
HS3	20	**7**	8	17
HS4	41	**8**	28	43
HS5	10	**0**	3	15
HS7	109	**32**	40	83

Table 9: *Performance comparison in terms of the number of soft constraint violations for the Beligannis data set [7] with three soft constraints.*

plemented by Beligiannis *et al.* [7], the particle swarm optimisation method employed by Tassopoulos *et al.* [36] and the simulated annealing approach used by Zhang *et al.* [40] to solve this problem. Particle swarm optimisation and simulated annealing perform better than GASPHH on the problem sets, with PSO outperforming all the approaches in solving these problems with the lowest constraint cost for all six problems.

The timetable used by the South African primary school is induced using a package, and is then manually changed to meet the hard and soft constraints. The current timetable used by the school does not meet all the double period requirements while both the GA [31] and GASPHH have produced feasible timetables. The success rate of the GA over thirty runs was 60% and GASPHH 93%. Furthermore, the quality of the timetables produced by GASPHH is better than the GA, and this was found to be significant at the 1% level.

The best timetable produced by the GA for the South African high school timetabling problem is a feasible timetable and has the same soft constraint cost, namely a cost of 2, as the timetable currently being used by the school. The soft constraint cost of the best timetable produced by GASPHH is also 2. However, the success rate of GASPHH over the thirty runs is 100% and 67% for the GA. The average quality of the timetables produced by GASPHH is 2.37 and for those produced by the GA 4.5. The better performance of GASPHH compared to the GA was found to be significant at the 1% level of significance.

GASPHH has performed well on all five school timetabling problems producing the best results when compared to other methods applied to the same problem instances. Furthermore, GASPHH has been able to generalize over the five different types of school timetabling problems.

7 Conclusion and future work

The study presented in this paper investigates the use of a selection perturbative hyper-heuristic in solving the school timetabling problem. A genetic algorithm selection perturbative hyper-heuristic is implemented and tested on five different types of school timetabling problems. This hyper-heuristic has produced feasible timetables for all problem instances and has provided a generalized solution to the school timetabling problem, performing better than other methods applied to the same set of problems. It is interesting to note that the genetic algorithm hyper-heuristic has performed better than the genetic algorithm applied directly to the solution space. It is hypothesized that small changes in the heuristic

space result in larger changes in the solution space than when searching the solution space directly, thereby facilitating quicker movement through the space and greater exploration of the solution space indirectly. This will be investigated further as part of future work. One of the disadvantages of hyper-heuristics is the higher runtimes as a result of having to construct a solution to evaluate each heuristic combination. This is especially so for population based methods as a population of heuristic combinations are evaluated over multiple generations. The use of high performance computing can help reduce the high runtimes associated with hyper-heuristics. Future work will also investigate the extension of the selection perturbative hyper-heuristic to educational timetabling in general, *i.e.* the hyper-heuristic will generalize over university examination timetabling, university course timetabling and school timetabling. Formal methods for identifying the appropriate set of low-level heuristics to be used could also be researched further.

Acknowledgements

The authors would like to thank the referees for their helpful comments and suggestions.

References

[1] Abdelmaguid TF, 2010, *Representations in Genetic Algorithm for the Job Shop Scheduling: A Computational Study*, Software Engineering and Applications, **3(12)**, pp. 1155–1162.

[2] Abramson D, 1991, *Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms*, Management Science, **37(1)**, pp. 98–113.

[3] Abramson D & Abela J, 1991, *A Parallel Genetic Algorithm for Solving the School Timetabling Problem*, Proceedings of the 15th Australian Conference: Division of Information Technology, pp. 1–11.

[4] Abramson D & Dang H , 1993, *School Timetable: A Case Study in Simulated Annealing*, Chapter 5, pp. 103–124 in *Applied Simulated Annealing*, Springer.

[5] Alvarez-Valdes R, Martin G & Tamarit JM, 1996, *Constructing Good Solutions for the Spanish School Timetabling Problem*, Journal of the Operational Research Society, **47(10)**, pp. 1203–1215.

[6] Avella P, D'Auria B, Salerno S & Vasil'Ev I, 2007, *A Computational Study of Local Search Algorithms for Italian High-School Timetabling*, Journal of Heuristics, **13(6)**, pp. 543–556.

[7] Beligiannis GN, Moschopoulos CN, Kaperonis GP & Likothanassis SD, 2008, *Applying Evolutionary Computation to the School Timetabling Problem: The Greek Case*, Computers and Operations Research, **35(4)**, pp. 1265–1280.

[8] Bufe M, Fisher T, Gubbels H, Hacker C, Hasprich O, Scheibel C, Weicker K, Weicker N, Wenig M & Wolfangel C, 2001, *Automated Solution of a Highly Constrained School Timetabling Problem Preliminary Results*, pp. 431–440 in *Applications of Evolutionary Computing*, Springer Berlin Heidelberg.

[9] Burke EK, Curtois T, Hyde M, Kendall G, Ochoa G, Petrovic S, Vazquez JA, 2009, *HyFlex: A Flexible Framework for the Design and Analysis of Hyper-Heuristics*, Proceedings of the Multidisciplinary International Scheduling Conference (MISTA 2009), Dublin, Ireland.

[10] Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Ozcan E & Qu R, 2013, *Hyper-Heuristics: A Survey of the State of the Art*, Journal of the Operational Research Society, **64(12)**, pp. 1695–1724, doi:10.1057/jors.2013.71.

[11] Burke E, Harte, Kendall G, Newall J, Ross P & Schulenburg S, 2003. *Hyper-Heuristics: An Emerging Direction in Modern Research*, Chapter 16, pp. 457-474 in Glover, Fred and Kochenberger, Gary A (Eds), *Handbook of Metaheuristics*, Springer.

[12] Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E & Woodard J, 2010, *A Classification of Hyper-Heuristic Approaches*, pp. 449–468 in *Handbook of Metaheuristics*, Springer.

[13] Cerdeira-Pena A, Carpente L, Farina A & Seco D, 2008, *New Approaches for the School Timetabling Problem*, Proceedings of the 7th Mexican International Conference on Artificial Intelligence, pp. 261–267.

[14] Colorni A, Dorigo M & Maniezzo V, 1998, *Metaheuristics for High School Timetabling*, Computational Optimization and Applications, **9(3)**, pp. 275-298.

[15] Filho GR & Lorena LAN, 2001, *A Constructive Evolutionary Approach to School Timetabling*, pp. 130-139 in *Applications of evolutionary computing*, Springer Berlin Heidelberg.

[16] Goldberg D, 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., Boston (MA).

[17] Han L & Kendall G, 2003, *An Investigation of a Tabu Assisted Hyper-Heuristic Genetic Algorithm*, Proceedings of the 2003 Congress on Evolutionary Computation CEC 2003, 8-12 December 2003, **3**, pp. 2230–2237.

[18] Koza JR, 1992, *Genetic Programming I: On the Programming of Computers by Means of Natural Selection*, MIT Press.

[19] McCollum B, McMullan P, Paechter B, Lewis R, Schaerf A, Digapsero L, Parkes AJ, Qu R & Burke EK, 2008, *Setting the research agenda in automated timetabling: the Second International Timetabling Competition*, INFORMS Journal of Computing, **22(1)**, pp. 120-130.

[20] Di Stefano C & Tettamanzi AGB, 2001, *An Evolutionary Algorithm for solving the School Time-Tabling Problem*, pp. 452–462 in *Applications of Evolutionary Computing*, Springer Berlin Heidelberg.

[21] 21. Ochoa G, Hyde M, Curtois T, Vazquez-Rodriguez JA, Walker J, Gendreau M, Kendall G, Mccollum B, Parkes AJ, Petrovic S & Burke EK, 2012, *HyFlex: A Benchmark Framework for Cross-Domain Heuristic Search*, pp. 136–147 in *Evolutionary Computation in Combinatorial Optimization*, SPringer Berlin Heidelberg.

[22] Pillay N, 2010, *A Study into the Use of Hyper-Heuristics to Solve the School Timetabling Problem*, Proceedings of SAICSIT 2010, Bela-Bela, South Africa, October 2010, pp. 258–264.

[23] Pillay N, 2011, *A Hyper-Heuristic Approach to Solving School Timetabling Problems*, Proceedings of MISTA 2011, Phoenix, Arizona, August 2011, pp. 628–632.

[24] Pillay N, 2011, *Evolving Heuristics for the School Timetabling Problem*, Proceedings of the 2011 IEEE Conference on Intelligent Computing and Intelligent Systems (ICIS 2011), Guangzhou, China, November 2011, **3**, pp. 281–286, IEEE Press.

[25] Pillay N, 2012, *Evolving Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem*, Journal of the Operational Research Society, **63(1)**, pp. 47–58.

[26] Pillay N, 2012, *Finding Solutions to Sudoku Puzzles Using Human Intuitive Heuristics*, South African Computer Journal, September 2012 **49**, pp. 25–34.

[27] Pillay N, 2014, *A Survey of School Timetabling Research*, Annals of Operations Research, **218(1)**, pp. 261–293.

[28] Post G, Ahmadi H, Daskalaki S, Kingston JH, Kyngas J, Nurmi C & Ranson D, 2012, *An XML Format for Benchmarks in High School Timetabling*, Annals of Operations Research, **194(1)**, pp. 385–397.

[29] Post G, Di Gaspero L, Kingston Jh, Mccollum B, Schaerf A, 2013, *The Third International Timetabling Competition*, Annals of Operations Research, February 2013, doi:10.1007/s10479-013-1340-5.

[30] Qu R, Burke EK, McCollum B, Merlot LTG & Lee SY, 2009, *A Survey of Search Methodologies and Automated System Development for Examination Timetabling*, Journal of Scheduling, **12(1)**, pp. 55-89.

[31] Raghavjee R & Pillay N, 2013, *A Study of Genetic Algorithms to Solve the School Timetabling Problem*, pp. 64–80 in *Advances in Soft Computing and Its Applications*, Springer Berlin Heidelberg.

[32] Randall M, 2000, *A General Meta-Heuristic Based Solver for Combinatorial Optimization Problems*, Computational Optimization and Applications, **20(2)**, pp. 185–210.

[33] Rahoual M & Saad R, 2006, *Solving Timetabling Problems by Hybridizing Genetic Algorithms and Tabu Search*, Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006), pp. 467–472.

[34] Rodriguez JAV, Petrovic S & Salhi A, 2007, *An Investigation of Hyper-Heuristics for Search Spaces*, Proceedings of the IEEE Congress on Evolutionary Computation CEC 2007, 25-28 September 2007, Singapore, pp. 3776-3783, IEEE.

[35] Smith KA, Abramson D & Duke D, 2003, *Hopfield Neural Networks for Timetabling: Formulations, Methods, and Comparative Results*, Computers and Industrial Engineering, Pergamon, **44(2)**, pp. 283–305.

[36] Tassopoulos IX & Beligiannis GN, 2012, *A Hybrid Particle Swarm Optimization Based Algorithm for High School Timetabling Problems*, Applied Soft Computing, **12(11)**, pp. 3472–3489.

[37] Tassopoulos IX & Beligiannis GN, 2012, *Solving Effectively the School Timetabling Problem Using Particle Swarm Optimization*, Expert Systems with Applications, **39(5)**, pp. 6029–6040.

[38] Valouxis C & Housos E, 2003, *Constraint Programming Approach for School Timetabling*, Computers and Operations Research, **30(10)**, pp. 1555–1572.

[39] Wilke P, Grobner M & Oster N, 2002, *A Hybrid Genetic Algorithm for School Timetabling*, pp. 455–464 in *AI 2002: Advances in Artificial Intelligence*, Springer Berlin Heidelberg.

[40] Zhang D, Liu Y, Hallah RM & Leung SCH, 2010, *A Simulated Annealing with a New Neighbourhood Structure Based Algorithm for High School Timetabling Problems*, European Journal of Operational Research, **203(3)**, pp. 550-558.