A Preliminary Study into the Use of an Evolutionary Algorithm Hyper-heuristic to Solve the Nurse Rostering Problem

Christopher Rae  
School of Mathematics, Statistics & Computer Science  
University of KwaZulu-Natal  
Pietermaritzburg, South Africa  
stingrae789@gmail.com

Nelishia Pillay  
School of Mathematics, Statistics & Computer Science  
University of KwaZulu-Natal  
Pietermaritzburg, South Africa  
pillayn32@ukzn.ac.za

Abstract—This paper reports on an initial attempt to solve the nurse rostering problem using an evolutionary algorithm selection perturbative hyper-heuristic. The main aim of this study is to get a feel for the potential of such a hyper-heuristic in solving the nurse rostering problem. This will be used to direct future extensions of this work. This study identifies low-level perturbative heuristics for this domain as well as a representation, initial population generation method, evaluation and selection methods, and genetic operator for the evolutionary algorithm hyper-heuristic. The approach was tested on six problems from the first international nurse rostering competition. The performance of the hyper-heuristic was found to be comparable to that of other methods applied to the same problems. The study has shown the potential of this approach and also identified future extensions of this work.

Keywords- nurse rostering; hyper-heuristics; perturbation; evolutionary algorithms

I. INTRODUCTION

The nurse rostering problem (NRP) is an NP-hard problem which involves the scheduling of nurses to shifts so as to ensure the problem constraints are met or the number of constraints violated is minimized. Various methods have been evaluated for solving this problem including tabu search, simulated annealing, integer programming, constraint programming and genetic algorithms [1] amongst others. A more recent approach for solving combinatorial optimization problems is hyper-heuristics which aims at producing a more general solution to a problem rather than the best result for one or two data sets [2]. There has not been much research into the use of hyper-heuristics for solving the nurse rostering problem. Furthermore, the use of an evolutionary algorithm (EA) selection perturbative hyper-heuristic which caters for both heuristic selection and move acceptance has not been conducted.

This paper reports on a preliminary study into the use of an evolutionary algorithm selection perturbative hyper-heuristic (EA-SPHH) for solving the nurse rostering problem. The aim of the study was to identify low-level perturbative heuristics for this domain and a representation, method for initial population generation, methods for evaluation and selection and genetic operators for the EA hyper-heuristic. An evaluation of the hyper-heuristic on six nurse rostering problems revealed the effectiveness of hyper-heuristics for this domain.

The following section describes the nurse rostering problem. An overview of hyper-heuristics and the application of hyper-heuristics to solve the nurse rostering problem is presented in section 3. Section 4 presents the EA-SPHH for solving the NRP. The experimental setup used to test the EA –SPHH is described in section 5. Section 6 discusses the performance of the EA hyper-heuristic in solving the NRP. An overview of the findings of the study and future work is provided in section 7.

II. THE NURSE ROSTERING PROBLEM

The nurse rostering problem is a personnel scheduling problem that falls under the category of NP-complete in difficulty. This field of study has potential real world applications due it being applicable to other personnel scheduling problems and as such has been studied for over 40 years.

The problem itself is to assign a set of shifts, defined as working periods, to the available nursing staff subject to the rules and regulations or better known as constraints of the employer and some of the preferences the nursing staff may have [3]. It has been said that this is the most challenging of the personnel scheduling problems [4]. Typically the nurse rostering problem defines the number of nurses and their skills, availability and shift preferences. These problem requirements are usually specified in terms of hard and soft constraints. Hard constraints must be met for a roster to be feasible. Soft constraints on the other hand may be violated as they are not mandatory but rather preferential requirements. Examples of hard constraints include a nurse may only work a single shift per day, a nurse must have a specific skill in order to work a certain shift and all shift requirements must be met. The aim is to minimize the soft constraint cost of a roster while keeping it free of hard constraint violations. A roster meeting all hard constraints is referred to as a feasible roster [5]. The quality of a roster is determined by its soft constraint cost. Soft constraints include minimum and maximum assignments per scheduling period, minimum or maximum consecutive days and nurse preferences like requests to work or not work on certain days amongst others.

Many approaches have been applied to this domain including constraint programming [6], expert systems [6],
genetic algorithms [3, 6, 7], heuristic techniques [6], integer programming [6], simulated annealing [6] and tabu-search [6, 8].

III. HYPER-HEURISTICS AND THE NURSE ROSTERING PROBLEM

Hyper-heuristics provide a more generalized solution to the problem and search a heuristic space instead of a solution space which is typical of most optimization techniques [9]. Hyper-heuristics are divided into four main categories, namely, selection constructive, selection perturbative, generation constructive and generation perturbative [2]. In this paper a selection perturbative hyper-heuristic is investigated for solving the nurse rostering problem. Selection perturbative hyper-heuristics explore a space of low-level perturbative heuristics. Perturbative low-level heuristic are essentially move operators, e.g. swap the shift of two nurses, and are used to improve the feasibility and/or quality of a roster.

All the research in the domain of hyper-heuristics for the nurse rostering problem have focused on selection perturbative hyper-heuristics. A few studies have combined the use of a selection perturbative hyper-heuristic with another optimization technique to solve the NRP. Bilgin et al. [10] used a random selection hyper-heuristic to solve the Belgian nurse rostering problem which used simulated annealing or great deluge for move acceptance. It was found that simulated annealing performed best. This study used 6 low-level heuristics to assign, delete and change shifts. Burke et al. [11] implemented a tabu-search selection perturbative hyper-heuristic to generate a nurse roster for a UK hospital. Nine low-level perturbation heuristics, which change nurse shifts, are used. Among the hybrid approaches is that employed by Bilgin et al. [12] which combines a greedy shuffle heuristic and a random selection hyper-heuristic with simulated annealing for move acceptance. This was one of 14 entries to the first international nurse rostering competition and placed within the top 5. This study used 12 low-level perturbative heuristics. The greedy shuffle is applied after the hyper-heuristic and swaps components of rosters for two nurses. Only swaps that produce feasible rosters of just as good or better quality are accepted. Bai et al. [13] use a hybrid of a genetic algorithm and a selective perturbative hyper-heuristic. The genetic algorithm uses the crossover and mutation operators on the solution space. The hyper-heuristic is embedded in the genetic algorithm and is used to improve each individual. Heuristics are selected based on their performance and simulated annealing is used for move acceptance. The low-level perturbative heuristics are those used in the study by Burke et al. [11] described above.

Evolutionary algorithms have previously been used to search heuristic spaces of both constructive and perturbative low-level heuristics. Han et al. [14, 15] employ the use of a genetic algorithm hyper-heuristic to explore a space of perturbative heuristics to solve the staff trainer scheduling problem. The work done by Pillay [16] examines the use of an evolutionary algorithm selection constructive hyper-heuristic to provide a solution to the examination timetabling problem. A similar approach is taken in this study in employing an evolutionary algorithm to search a space of low-level perturbative heuristic combinations to solve the nurse rostering problem.

IV. EVOLUTIONARY ALGORITHM HYPER-HEURISTIC

This section presents the evolutionary algorithm hyper-heuristic used to solve the nurse rostering problem. The algorithm is generational with a set number of generations being implemented. An initial population is firstly created. This population is then iteratively refined during each generation by means of evaluation, selection and regeneration (using genetic operators). The overall algorithm implemented is outlined in Figure 1. The following sections describe each of these processes.

Create initial population
Repeat
Evaluate population
Select parents using tournament selection
Apply genetic operators
Until a maximum number of generations has been reached or the solution has been found

Figure 1. Generational EA implemented

A. Chromosome representation and initial population generation

An individual in the population is represented by a string formed by characters representing low-level perturbative heuristics. Various low-level heuristics, based on those reported in the literature and human intuition, were tested during trial runs and the following performed well and were hence included in the heuristic set:

- Swap day first improving or equal accept first (i) - Selects two nurses and a day randomly. The shifts on the day are swapped for both nurses. Accepts the first equal or improving solution.
- Swap day first improving or equal accept best found (s) - Selects two nurses and a day randomly. The shifts on the day are swapped for both nurses. Accepts all improving or equal but will keep trying until a limit of tries is reached.
- Swap subset of days first improving or equal accept first (y) - Selects two nurses randomly and then selects a random number of days on which the shifts for those nurses are swapped. Accepts the first equal or improving solution.
- Swap subset of days improving or equal accept best found (x) - Selects two nurses randomly and then selects a random number of days on which the shifts for those nurses are swapped. Accepts all improving or equal but will keep trying until a limit of tries is reached.
Each element of the initial string is randomly selected from the set of low-level perturbative heuristics. A character is used to represent each heuristic. These have been specified in brackets as part of the definition of each heuristic above. A limit, which is a function of the number of shifts to be assigned, is placed on the length of each individual. An example of an individual is $sssttttvvc$.

B. Evaluation and Selection

Prior to the application of the EA-SPHH an initial roster is created by randomly allocating shifts to nurses. Each element of the population is evaluated by using it to improve a roster. For example $sttv$ will be used to improve a roster applying each of the low-level heuristics in order to the roster. In successive generations the best roster created from the previous generations is retained and each individual is applied to this roster. Thus, a best roster is stored and adapted throughout a run of $n$ generations. This approach is different from that taken in similar studies employing evolutionary algorithm hyper-heuristics for rostering. The fitness of an individual is the sum of the hard and soft constraint violations of the roster adapted using the individual and this value is minimized.

Tournament selection is used to choose parents of the next generation. This selection method returns the fittest individual in a tournament of $t$ randomly selected individuals.

C. Genetic Operators

The population of each generation is created by applying the mutation, crossover and permutation operators to selected parents. Trial runs indicated the benefit of the permutation operator in addition to mutation and crossover. This was verified by performing ten runs with and without the use of the permutation operator. This is illustrated in Table 1 below. The following information is displayed:

- 1 - Permutation rate
- 2 - Mutation rate
- 3 - Crossover rate
- 4 - Generation on which the solution with the minimum soft constraint cost was found.
- 5 - Time taken, in seconds, to find a solution with the minimum cost
- 6 - Percentage of runs producing the minimum soft constraint cost.

Table 1. Comparison of Genetic Operator Application Rates

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>30%</td>
<td>60%</td>
<td>10</td>
<td>250.62</td>
<td>70%</td>
</tr>
<tr>
<td>0%</td>
<td>40%</td>
<td>60%</td>
<td>12</td>
<td>324.06</td>
<td>50%</td>
</tr>
<tr>
<td>0%</td>
<td>30%</td>
<td>70%</td>
<td>17</td>
<td>551.52</td>
<td>50%</td>
</tr>
</tbody>
</table>

There is no limit placed on the size of the offspring created. The mutation operator randomly chooses a mutation point in an individual and replaces the low-level heuristic at this point with a randomly chosen heuristic. An example of mutation is illustrated in Figure 2. In this example the mutation point has randomly selected to be 4 and the heuristic at this point has been replaced by $t$.

![Figure 2. Example of mutation](image)

The crossover operator combines two individuals selected using tournament selection. This is achieved by selecting two random points, one in each individual. The substrings created by the crossover points are swapped to create two offspring. The fitter of the two offspring is returned as the result of the process. An example of crossover is depicted in Figure 3.

---

1 Is the best known soft constraint cost for each problem.
Selected individual one:
\[
\text{s s t v v u v t t s}
\]

Selected individual two:
\[
\text{s t s b t u v}
\]

First child:
\[
\text{s s t t u v}
\]

Second child:
\[
\text{s t s b t v v u v t t s}
\]

Evaluate each individual and return the best newly created individual.

The permutation operator performs a shuffle of the elements of the string thereby changing the position of each heuristic. This process is illustrated in Figure 4 where it can be seen that some elements have been moved randomly e.g. t is now at position 3.

The EA hyper-heuristic was evaluated on six randomly chosen problems that were used for the Sprint track of the first International Nurse Rostering competition [8]. Six problems were randomly selected for testing purposes. The details for these problems are listed in Table 2.

<table>
<thead>
<tr>
<th>First International Nurse Rostering Problem Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Best Known Result</strong></td>
</tr>
<tr>
<td><strong>Sprint Early 01</strong></td>
</tr>
</tbody>
</table>

Due to the stochastic nature of genetic algorithms, ten runs, each using a different random number generator seed, was performed for each problem. The parameter values used by the evolutionary algorithms are listed in Table 3. These have been obtained empirically by performing trial runs.

<table>
<thead>
<tr>
<th>Table 3. EA Parameter values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>Population Size</td>
</tr>
<tr>
<td>Initial String Limit</td>
</tr>
<tr>
<td>Tournament size</td>
</tr>
<tr>
<td>Crossover percentage</td>
</tr>
<tr>
<td>Mutation percentage</td>
</tr>
<tr>
<td>Permutation percentage</td>
</tr>
</tbody>
</table>

The EA-SPHH was implemented in Java 1.7 and all simulations were run on an Intel I7 (34 GHz) with 8 GB of RAM and Windows 7.

VI. RESULTS AND DISCUSSION

EA-SPHH was able to produce feasible rosters for all the problems it was evaluated on. Table 4 displays the minimum and average soft constraint cost and runtimes over ten runs for each problem.

<table>
<thead>
<tr>
<th>Table 4. Costs and Runtimes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>SprintEarly01</td>
</tr>
<tr>
<td>SprintEarly06</td>
</tr>
<tr>
<td>SprintEarly02</td>
</tr>
<tr>
<td>SprintHidden02</td>
</tr>
<tr>
<td>SprintHidden04</td>
</tr>
<tr>
<td>SprintLate10</td>
</tr>
</tbody>
</table>

For each problem the individual producing the best solution quality roster differed from one run to another. Different combinations also proved to be more effective for the different problems.

Hyper-heuristics aim to generalize well over a set of problems instead of producing the best solution for one or two problems. Furthermore, the runtimes of hyper-heuristics can be expected to be higher than standard optimization given that the evaluation of each heuristic combination involves creating a solution using the combination. However, for completeness and to get a feel of the effectiveness of the EA hyper-heuristic, the results obtained by the hyper-heuristic was compared to the methods cited in
the literature as producing the best results for the same set of problems. These include:
1. The perturbative selection hyper-heuristic using random selection and simulated annealing [10].
2. The ejection chain and branch and price algorithm hybrid implemented by Burke et al. [17].
3. The constraint optimization approach taken by Konobe [18].
4. Integer programming hybrid [19].
5. Adaptive local search [20].
6. Integer programming [21].
7. Harmony search [22].
8. Adaptive neighbourhood search [23].

The best soft constraint cost produced by these methods and EA-SPHH are listed in Table 4. Note that a hyphen indicates that the method was not able to find a feasible solution. The best results are highlighted in bold.

<table>
<thead>
<tr>
<th></th>
<th>Sprint Early 01</th>
<th>Sprint Early 06</th>
<th>Sprint Early 02</th>
<th>Sprint Hidden 02</th>
<th>Sprint Hidden 04</th>
<th>Sprint Late 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA-SPHH</td>
<td>56</td>
<td>58</td>
<td>54</td>
<td>32</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Lowe [10]</td>
<td>57</td>
<td>59</td>
<td>54</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lowe [19]</td>
<td>56</td>
<td>58</td>
<td>54</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lowe [20]</td>
<td>56</td>
<td>58</td>
<td>54</td>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lowe [21]</td>
<td>56</td>
<td>58</td>
<td>54</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lowe [22]</td>
<td>62</td>
<td>63</td>
<td>60</td>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lowe [23]</td>
<td>56</td>
<td>58</td>
<td>54</td>
<td>32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is clear from Table 3 that not all methods were able to find feasible solutions for all of the problems. EA-SPHH has performed well as it was firstly able to produce feasible solutions for all six nurse rostering problems. Furthermore, the best soft constraint costs obtained by the EA-SPHH are either the known minimum or close to the know minimum for all six problems.

VII. CONCLUSION AND FUTURE WORK

This paper reports on a preliminary study conducted to ascertain the potential of an evolutionary algorithm selection perturbative hyper-heuristic in solving the nurse rostering problem. The study identified the low-level perturbative heuristics for this problem and the overall architecture for the EA hyper-heuristics. The study introduced a new genetic operator, namely, permutation, which improved the performance of the EA for this domain. The study also revealed the effectiveness of a shared memory approach which involved maintaining the best timetable over a run and improving this timetable during each generation. The EA-SPHH was evaluated on six nurse rostering problems. EA-SPHH was able to produce feasible rosters for all six problems. Furthermore, the soft constraint cost of the rosters were comparable to and in some cases better than other methods applied to the same set of problems.

This study has clearly shown the potential of the EA-SPHH in solving the nurse rostering problem. Future work will involve conducting an analysis of the low-level heuristics used to evaluate their performance and will possibly examine the use of other perturbation heuristics such as allocate and de-allocate as well as testing the EA-SPHH on additional nurse rostering problems.

REFERENCES


