Security personnel routing and rostering: a hyper-heuristic approach

Mustafa Misir[1,2], Pieter Smet[1], Katja Verbeeck[1,2] and Greet Vanden Berghe[1,2]

[1] KAHO Sint-Lieven, CODeS, Gebroeders De Smetstraat 1, 9000 Gent, Belgium
[2] Katholieke Universiteit Leuven Campus Kortrijk, Computer Science and Information Technology, Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
{Mustafa.Misir, Pieter.Smet, Katja.Verbeeck, Greet.Vandenberghe}@kahosl.be

Abstract. In the present study, a large scale, *structured* problem regarding the routing and rostering of security personnel is investigated. Structured problems are combinatorial optimization problems that encompass characteristics of more than one *known* problem in operational research. The problem deals with assigning the available personnel to visits associated with a set of customers. This objective just described, reflects the rostering characteristic of the problem. In addition, the different geographic locations of the customers indicate the requirement of routing. A new benchmark dataset for this complex problem is presented. A group of high-level problem-independent methods, i.e. hyper-heuristics, is used to solve this novel problem. The performance and behaviour of different hyper-heuristics for the presented benchmark dataset are analysed.

Keywords: vehicle routing, personnel rostering, hyper-heuristic

1. Introduction

Security personnel routing and rostering deals with the complex problem of assigning security tasks at different locations to the members of staff. Task duration and staff requirements vary and so do the skills and contracts of the personnel.

The problem presented can be considered a combination of two well-known, hard problems: the vehicle routing problem with time windows (VRPTW) (Cordeau et al. 2001) and the personnel rostering problem (Burke et al. 2004). More specifically, full attention is given to the multi-depot variant of the VRPTW (MDVRPTW) allowing the model to come closer to the real-world situation. The MDVRPTW has received limited attention in the literature. Cordeau et al. 2001 present results for a tabu search algorithm and introduce new benchmark instances for the MDVRPTW. Polacek et al. describe the application of a variable neighbourhood search algorithm and report an improvement for 10 of the 20 Cordeau benchmark instances. Ostertag et al. 2009 use a memetic algorithm to solve a large real world MDVRPTW, with up to 1848 customers to be serviced.

In contrast to the MDVRPTW, personnel rostering has received ample attention over the last decades. Burke et al. 2003 present the application of a tabu search hyper-

heuristic to a nurse rostering problem. The hyper-heuristic produces schedules of similar quality as those produced by a (tailor-made) genetic algorithm. The authors illustrate the high level of generality that a hyper-heuristic provides by applying it to another scheduling problem. Bilgin et al. 2009 present a hyper-heuristic approach to a nurse rostering problem in Belgian hospitals. For 16 out of 18 problem instances their hyper-heuristics perform significantly better than a variable neighbourhood search algorithm, for the two remaining instances their performance is similar.

The home care scheduling problem (HCSP) exhibits similar characteristics as the problem investigated here. The difference lies in the limited complexity of the personnel rostering aspects in the HCSP. For example, typically, the scheduling horizon is set to one (Bertels and Fahle 2006) (or five (Begur et al. 1997)) days.

In the literature, hyper-heuristics have been investigated under two main types: *selection hyper-heuristics* and *generation hyper-heuristics* (Burke et al. 2010). Selection hyper-heuristics are composed of different components to manage a set of search mechanisms. Choice function (Cowling et al. 2001), reinforcement learning (Nareyek 2003, Ozcan et al. 2010), case-based reasoning (Burke et al. 2002), simulated annealing (Bai et al. 2005, Burke et al. 2010) and genetic algorithms (Han and Kendall 2003) are some example techniques that have been employed to build effective selection hyper-heuristics. For generation hyper-heuristics, the idea is to automatically generate low-level search strategies. Here, genetic programming (Burke et al. 2006) is the preferred approach. More details about hyper-heuristics are available in Burke et al. 2010, Burke et al. 2009.

In the present study, the problem of routing and rostering security personnel is studied. Various selection hyper-heuristics are applied to a set of real-world problem instances. In the next section, the details of the problem are explained. In Section 3, the details of the solution strategy with applied hyper-heuristics are presented. Experimental results are analysed in Section 4. In the last section, concluding remarks and future ideas are discussed.

2. Problem Description

Let $G = (V, A)$ be a complete graph with vertices $V = \{v_1, \ldots, v_d, v_{d+1}, \ldots, v_n\}$ and arcs $A = \{(v_i, v_j) : v_i, v_j \text{ in } V, i \neq j\}$. The nodes $D = \{v_1, \ldots, v_d\}$ represent the depots, and nodes $N = V \setminus D$ represent the jobs that need to be performed by the security personnel. These jobs will be referred to as *visits*.

With each depot v_j in D, a number of personnel k_j is associated, who start and end their tours in the depot v_j. Let $P_j = \{p_1, \ldots, p_k\}$ be the set of security guards at each depot. In the problem presented, depots are associated with homes of security guards and each employee leaves for the first visit on his or her route straight from his or her home. Typically, $k_j = 1$ for every v_j in D, which means that there is only one employee at each depot. However, other situations are possible, when e.g. two colleagues live at the same place.

With each security guard p_i in P_j, a set of skills $S_i = \{s_1, \ldots, s_m\}$ is associated. Furthermore, a list of regions is specified in which a security guard is allowed to perform visits. If a security guard is assigned to a visit that is located outside the

guard's allowed regions, a penalty will be incurred. Finally, a contract describing a set of workforce related constraints is specified for each security guard. Table 1 shows an overview of these constraints.

Table 1. Workforce related constraints.

Maximum number of consecutive working days	6
Maximum consecutive working time per day (minutes)	720
Maximum working time per week (hours)	37
Maximum working time per month (hours)	175 or 190
Minimum rest time between two working days (hours)	5
Maximum number of consecutive working weekends	2
Preferred number of days worked per week	5

All security guards are assumed available 24 hours per day. By defining the constraint that limits the maximum working time per day, they can only be assigned for 720 minutes in that period. Every security guard should return to his or her depot before this predetermined time limit is reached. Note that the working time per day is continuous. Apart from the legal workforce related constraints, which are identical for every employee, the available security personnel can be heterogeneous in any of the above described characteristics.

Every visit v_i in N can have an earliest and latest start date $[e_i, l_i]$, with $l_i \geq e_i$, as well as a time window for each day, defined by a start and end time $[t^s_i, t^e_i]$. Furthermore, a duration d_i is defined, so that $t^e_i \geq t^s_i + d_i$. Note that the earliest and latest start dates lie anywhere between the start and the end of the scheduling period. Lastly, a set of required skills, $R_i = \{s_1,\ldots,s_m\}$, is specified.

The benchmark problems used in this paper are based on real world data by the company Fascinating IT Solutions (www.fit.be). The data describe the characteristics of the security personnel and details about as to when visits have been executed. This means that every visit in the provided data has a *tight* time window, i.e. $l_i = e_i$ and $t^e_i = t^s_i + d_i$. In order to diversify the available benchmark problems, a second set of instances was created. In this second set, the parameters relating to the time windows, $[e_i, l_i]$ and $[t^s_i, t^e_i]$, are adjusted in such a way that for some visits the time window is no longer tight. This is done by relaxing the time window with a value $r_i = \alpha \, . \, d_i$, with α having a uniformly distributed chance of being either 0.5 or 1. Half of the resulting value r_i is then added to t^s_i, and the other half to t^e_i. New values for e_i and l_i are determined by adding a randomly chosen number of days from a uniform distribution between 0 and 4 to the original values of e_i and l_i. If, by extending e_i (or l_i) the bounds of the scheduling period are exceeded, the earliest (or latest) start date is removed for that visit.

For every benchmark problem, the length of the scheduling period is set to 31 days; the number of skills is 16 and the number of regions is 10. An overview of specific properties is given in Table 2. To give other researchers the opportunity to investigate new solution approaches for this problem, the benchmark instances have been made publicly available at http://allserv.kahosl.be/~pieter/securityguards.html.

The goal is to construct a set of routes in which every security guard starts from his or her depot and returns to the same depot. All visits in N need to be assigned to a security guard, and the start time of service has to be determined. The constructed

schedule should minimize the number of required employees as well as the required travel and waiting time for each employee. Furthermore, several requirements of both security personnel and customers are considered: violation of a visits' time window, violation of required skills, allowed regions and of the legal workforce related constraints. A weighted objective function is used to evaluate a candidate solution, in which the incurred penalties scale linearly with the amount of violation.

Table 2. Properties of the benchmark problems.

Name	Nr. of guards	Nr. of visits
district0	62	1560
district1	348	4217
district2	120	1714
district3	113	2193
district4	192	5252
district5	389	5139

3. Selection Hyper-heuristics

A traditional selection hyper-heuristic operates on a set of low-level heuristics for indirectly solving a problem instance. A low-level search heuristic can be any method that is capable of finding a solution or constructing a part of a solution for a problem instance. A selection hyper-heuristic chooses a heuristic and applies it to a solution. This selection procedure is carried out by a *heuristic selection* mechanism. After applying the selected heuristic, the constructed solution is examined based on its problem-independent characteristics, e.g. its quality, by a *move acceptance* mechanism. These consecutive operations are repeated at each decision step and terminated until a given stop criterion is reached.

A single-point search perturbative selection hyper-heuristic framework is used in this paper. This framework operates on a single complete solution and executes perturbations on this solution. The employed hyper-heuristic components are discussed in the following subsections.

3.1 Heuristic selection

In the present study, the selection operation is performed by two heuristic selection mechanisms, namely simple random (SR) heuristic selection and adaptive dynamic heuristic set (ADHS). SR is a naive but effective method that randomly chooses heuristics. ADHS behaves in a more selective way. The aim of ADHS is to determine the best heuristic subset for the current search region. In Algorithm 1, the details of ADHS are presented. In the given pseudo-code, a phase refers to a number of iterations for measuring the performance of the heuristics. *Tabu* is a status type for heuristics that are excluded for a number of phases (d). This value is calculated as

$d = \sqrt{2n}$, whereby n refers to the number of heuristics. Whenever a heuristic is not tabu anymore, its status is changed to *non-tabu* and added back to the heuristic set.

Algorithm 1: ***ADHS***

1. Check performance (p_i) of the heuristics during a phase
2. Assign a score (QI_i *in {1,2.,…,n}*) to each heuristic (LLH_i) based on its performance, with respect to the pre-determined performance metric
3. Exclude the heuristics having smaller scores than the average (*avg*) of all scores

$$avg = \left\lfloor \left(\sum_{i}^{n} QI_i \right) / n \right\rfloor$$

4. Change the status of the excluded heuristics to *tabu*
5. Change the status of the heuristics whose exclusion finishes to *non-tabu*

Equation (1) shows the performance metric used to determine *QI* values. Each element which is multiplied by a weight (w_i) refers to a performance related feature. The first performance feature is used to measure the capability of finding new best solutions. For the calculation, the number of new best solutions ($C_{p,best(i)}$) discovered during a phase (p) by heuristic i is used together with the speed of the heuristic. $C_{p,equal(i)}$ and $C_{p,moves(i)}$ are the counters showing the number of equal quality solutions generated and the number of moves performed by heuristic i during a phase. The outcome of this feature shows a value related to the number of new best solutions that can be found by the heuristic until the search is terminated. The second and third performance features show the fitness improvement ($f_{p,imp(i)}$) and fitness worsening ($f_{p,wrs(i)}$) per spent execution time ($t_{p,spent(i)}$) by heuristic i during a phase. The last two sub-performance metrics refer to the overall improvement ($f_{imp(i)}$) and worsening ($f_{wrs(i)}$) yielded by heuristic i per spent execution time ($t_{spent(i)}$) until that time. t_{remain} indicates the remaining execution time. In addition, the weights for the performance features are set as $w_1 >> w_2 >> w_3 >> w_4 >> w_5$. This allows the different features to be used according to their importance. In particular, if a heuristic discovers a new best solution, then there is no need to check other features to determine its *QI* value.

$$\begin{aligned} p_i = \quad & w_1\left(\left(C_{p,best}(i) + (C_{p,equal}(i) / C_{p,move}(i))\right)^2 \times \left(t_{remain} / t_{p,spent}(i)\right)\right) \times b + \\ & w_2\left(f_{p,imp}(i) / t_{p,spent}(i)\right) - w_3\left(f_{p,wrs}(i) / t_{p,spent}(i)\right) + \\ & w_4\left(f_{imp}(i) / t_{spent}(i)\right) - w_5\left(f_{wrs}(i) / t_{spent}(i)\right) \end{aligned} \tag{1}$$

$$b = \begin{cases} 1, & if \sum_{i=0}^{n} C_{p,best}(i) > 0 \\ 0, & otherwise \end{cases}$$

Tabu duration adaptation: The number of phases denoting the amount of time that a heuristic will be excluded (i.e. the time a heuristic is tabu), is determined in advance as mentioned above. Whenever the punishment of the excluded heuristic expires, it re-joins the active heuristic set. However, it is obvious that when an unsatisfactory

heuristic repeatedly re-joins the active set, the effectiveness of the hyper-heuristic can decrease. Therefore, instead of fixing the tabu duration for each heuristic, this value is updated when necessary. The adaptation method increments the tabu duration of a heuristic if it is excluded again after the phase in which it re-joined the heuristic set. It is incremented until the corresponding value reaches $2d_{initial}$. If this heuristic is not prohibited after a phase, then its tabu duration is set to $d_{initial}$ again.

Phase length adaptation: The phase length (ph_{length}) is calculated using a predefined value, i.e. ph_{factor}, as $\mathrm{ph}_{\mathrm{factor}} \times d, d = \sqrt{2n}$. In this formula, n refers to the number of heuristics in the current heuristic set. That is, the size of the heuristic set affects ph_{length}. The other adaptation element is associated with the speed of the non-tabu heuristics. At the end of each phase, the speed of performing one move based on the average speed of the non-tabu heuristic is used to determine the ph_{length}. If the current heuristic set is slow, then ph_{length} is assigned a smaller value.

Learning automata based selection: For efficiently using an elite heuristic subset, the performance of the heuristics with respect to their improvement capabilities during each phase is taken into account. For this purpose, heuristics are selected based on their performance related probabilities maintained by learning automata [20]. In particular, a linear reward-penalty update scheme is used. The corresponding probability vector is reset at the beginning of each phase.

3.2 Move acceptance

For the experiments, five move acceptance mechanisms are used: adaptive iteration limited list-based threshold accepting (AILLA), improving or equal (IE), simulated annealing (SA), great deluge (GD), late acceptance (LATE). The implementations of SA and GD are from [19]. The pseudo-code of LATE and AILLA are given in Fig. 1 and Fig. 2 respectively.

Input – *L*: list length (*l=10*)
Variables – *I*: number of iterations, *v*: selected index from the list
Initialize – Set all the list elements to the initial fitness

1. *v = I%L;*
2. **if** $f(S') \leq f(S_v)$ **then**
3. $S \leftarrow S'$;
4. $f(S_v) = f(S)$;
5. **end**
6. *I = I+1;* *//increase the number of iterations by 1*

Fig. 1. Pseudo-code of the late acceptance.

Input – *k*: initial iteration limit (*k=150*), *l*: list length (*l=10*), K: iteration limit for adaptation (*K=5k=750*)

```
Variables – i: threshold index for the list, w_iterations: the number of consecutive
            worsening moves, adapt_iterations: the number of iterations before increasing
            the threshold level
Initialize – Fill the best fitness list with the initial fitness
1.  if adapt_iterations ≥ K then
2.      if i < l-1 then
3.          i++;                        //increase the threshold value
4.      end
5.  end
6.  if f(S) < f(S') then
7.      S ← S';
8.      w_iterations = 0;
9.      if f(S') < f(S_b) then
10.         i = 1;
11.         S_b ← S';
12.         w_iterations = adapt_iterations = 0;
13.         best_list.remove(last);     //remove the last element in the list
14.         best_list.add(0,f(S_b));    //add new best solution to the beginning of the list
15.     end
16. else if f(S') = f(S) then
17.     S ← S';
18. else
19.     w_iterations++;
20.     adapt_iterations++;
21.     if w_iterations ≥ k and f(S') ≤ best_list(i) then
22.         S ← S';
23.         w_iterations = 0;
24.     end
25. end
```

Fig. 2. Pseudo-code of the adaptive iteration limited list-based threshold accepting.

Each of the used move acceptance mechanisms accepts solutions of better or equal quality. For diversifying the search process, SA accepts worsening solutions based on a probability calculated as shown in equation (2). GD accepts worsening solutions using a threshold value determined based on the quality of the initial solution ($f_{initial}$) as well as a linearly decreasing ratio for the remaining execution time (t_{remain}) over the total execution time (t_{total}) as illustrated in equation (3). LATE maintains the history of previously visited solutions for deciding about new solutions. Similarly, AILLA also maintains a history. However, it waits for a number of iterations denoting consecutively visited solutions (k) before diversifying the search. To indicate the hardness of finding new best solutions, k is updated whenever a new best solution is found. The simple update process is depicted in equation (4).

$$\exp\left[\left(f_{current} - f_{new}\right) \times \left(t_{total} / t_{remain}\right)\right] \tag{2}$$

$$f_{initial} \times \left(t_{remain} / t_{total}\right) \tag{3}$$

$$k = \left\{ \begin{array}{l} \left(\left((l-1) \times k + iter_{elapsed} \right) / l, \quad if \quad cw = \lfloor iter_{elapsed} / k \rfloor = 0 \right) \\ \left((l-1) \times k + \sum_{i=0}^{cw} k \times 0.8^i \times (t_{total} - t_{elapsed}) / t_{total} \right) / l, \quad otherwise \end{array} \right\} \tag{4}$$

4. A Two-phase Approach

A two-phase strategy involving the application of hyper-heuristics was designed to solve the routing and rostering problem. In the first phase, the best order of visits and assignments of these visits to the available personnel are searched for. The start time of each visit is determined by its earliest possible starting time as well as by the availability of the corresponding personnel. If no change occurs in the rest of the route, the start time remains unchanged during this first phase. The search is performed on a smaller space compared with the search space of the original problem.

In the first phase, the hyper-heuristic operates on a set of simple low-level heuristics (*LLH*s). The idea behind utilising such simple heuristics is to show that good quality solutions can be reached by simple operations instead of complex, highly problem-dependent moves. These heuristics are the following:

- LLH_0: swap two randomly selected visits between two randomly selected routes
- LLH_1: move the most conflicting visit in a randomly selected route to the best position in another randomly selected route.
- LLH_2: move a randomly selected visit from a randomly selected route to another randomly selected route. The best location to move the visit to is determined based on the quality of the route.
- LLH_3: move a randomly selected visit from a randomly selected route to another randomly selected route
- LLH_4: swap two randomly selected visits in a randomly selected route
- LLH_5: reverse a number randomly selected consecutive visits in a randomly selected route
- LLH_6: move a randomly selected visit in a randomly selected route
- LLH_7: scramble randomly selected consecutive visits in a randomly selected route
- LLH_8: exchange two randomly selected consecutive visits belonging to two randomly selected routes
- LLH_9: move a number of randomly selected consecutive visits from a randomly selected route to another randomly selected route

The second phase is dedicated to improving the solution by performing simple changes on visiting times. A time shifting parameterised heuristic is employed with four different parameter settings. They consist of sliding a visit +10 minutes, -10 minutes, +20 minutes and -20 minutes. In the experiments, this phase is handled by an SR-IE hyper-heuristic.

5. Experiments

Each hyper-heuristic was tested on 12 problem instances using a Pentium Core 2 Duo 3 Ghz with 3 GB memory. Each test was repeated 10 times per instance. The execution time was limited to 10 minutes per run. As described in Section 2, some of the time windows are relaxed in instances district0 to district5. In addition, the experiments were also performed on the same instances with tight time windows (instances district0-etw to district5-etw).

Table 3 shows the results of the hyper-heuristics with SR as selection mechanism. Results based on the best solutions at each run indicate that SR-AILLA, SR-SA and SR-IE show similar performance. SR-LATE performs better than SR-GD. However, they both perform significantly worse than the first three hyper-heuristics (based on the Wilcoxon signed-rank test with 95% confidence level.)

Table 3. The average fitness values with standard deviation for hyper-heuristics with SR.

Instances	SR-AILLA		SR-SA		SR-IE	
	AVG	STD	AVG	STD	AVG	STD
district0	3,44E+08	2,66E+07	3,16E+08	1,88E+07	3,39E+08	4,73E+07
district1	2,84E+05	1,63E+04	2,82E+05	1,31E+04	2,87E+05	1,71E+04
district2	1,16E+05	2,13E+05	1,72E+05	3,71E+05	5,31E+04	4,76E+04
district3	8,70E+04	2,23E+04	1,16E+05	1,10E+05	8,76E+04	2,39E+04
district4	1,17E+08	2,19E+07	1,14E+08	2,33E+07	1,27E+08	2,43E+07
district5	4,26E+05	7,62E+04	1,05E+07	3,16E+07	4,49E+05	1,24E+05
district0-etw	1,54E+08	5,63E+07	1,41E+08	2,78E+07	1,45E+08	3,75E+07
district1-etw	1,05E+05	1,10E+04	1,31E+05	8,53E+04	1,02E+05	6,07E+03
district2-etw	6,85E+04	5,92E+04	1,31E+05	1,88E+05	6,33E+04	4,38E+04
district3-etw	1,55E+05	2,13E+05	6,55E+04	6,57E+04	8,18E+04	6,71E+04
district4-etw	6,20E+07	9,19E+06	6,31E+07	1,06E+07	6,09E+07	1,01E+07
district5-etw	4,78E+05	4,82E+05	3,88E+05	1,11E+05	3,54E+05	1,33E+05
	SR-LATE		SR-GD			
	AVG	STD	AVG	STD		
district0	8,06E+08	5,79E+07	1,65E+09	9,57E+07		
district1	2,89E+05	1,33E+04	3,38E+08	1,83E+08		
district2	4,55E+05	1,11E+06	5,07E+08	1,43E+08		
district3	3,24E+05	7,33E+05	3,99E+07	7,00E+06		
district4	6,92E+08	1,12E+08	1,00E+09	1,52E+08		
district5	2,37E+08	2,36E+08	3,52E+09	5,51E+08		
district0-etw	4,47E+08	6,12E+07	1,15E+09	1,71E+08		
district1-etw	1,28E+05	4,05E+04	2,01E+08	1,36E+08		
district2-etw	1,15E+05	1,20E+05	4,22E+08	9,50E+07		
district3-etw	1,61E+05	4,01E+05	4,07E+07	2,84E+07		
district4-etw	2,70E+08	6,51E+07	5,70E+08	9,23E+07		
district5-etw	1,01E+08	9,45E+07	2,07E+09	4,24E+08		

3.1 Adaptive dynamic heuristic set strategy

Table 4 presents the performance of the set of significantly better hyper-heuristics from Table 3 with the ADHS selection mechanism instead of SR. These results indicate that ADHS provides further improvements over SR.

Table 4. The average fitness values with standard deviation for the top hyper-heuristics from Table 3 with ADHS.

Instances	ADHS-AILLA		ADHS-SA		ADHS-IE	
	AVG	STD	AVG	STD	AVG	STD
district0	3,39E+08	2,66E+07	3,18E+08	1,81E+07	3,12E+08	1,27E+07
district1	2,79E+05	6,37E+03	2,82E+05	1,26E+04	2,83E+05	1,21E+04
district2	5,87E+04	6,15E+04	3,83E+04	2,85E+04	7,56E+04	1,44E+05
district3	8,62E+04	2,72E+04	8,87E+04	1,67E+04	1,03E+05	4,36E+04
district4	6,02E+07	1,45E+07	5,75E+07	1,09E+07	5,73E+07	1,39E+07
district5	4,10E+05	7,44E+04	5,09E+05	1,96E+05	4,99E+05	1,35E+05
district0-etw	1,39E+08	1,61E+07	1,24E+08	1,38E+07	1,21E+08	9,22E+06
district1-etw	1,02E+05	7,86E+03	1,28E+05	7,36E+04	1,22E+05	3,45E+04
district2-etw	2,61E+04	1,83E+04	4,43E+05	1,24E+06	2,59E+04	1,14E+04
district3-etw	4,28E+04	2,64E+04	4,22E+04	2,61E+04	7,34E+04	7,40E+04
district4-etw	3,34E+07	7,81E+06	3,34E+07	7,55E+06	3,51E+07	1,00E+07
district5-etw	3,70E+05	8,00E+04	4,75E+05	2,48E+05	3,42E+05	4,98E+04

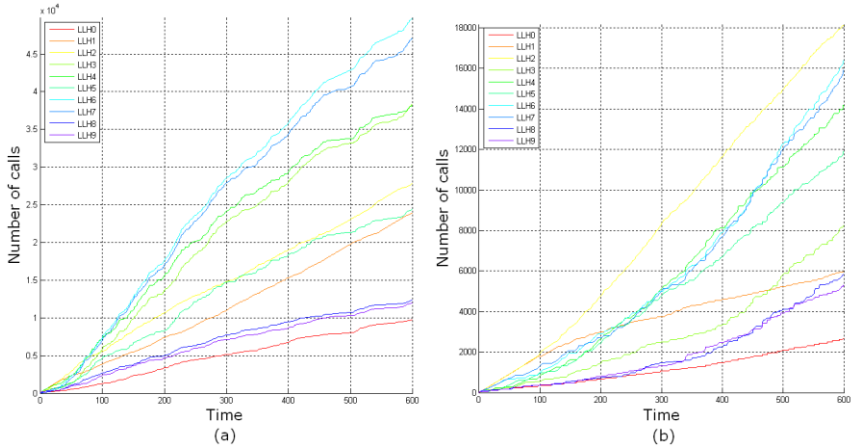

Fig. 3. The number of moves performed by each heuristic during the run by ADHS-AILLA on (a) district2-etw run 3, (b) district4-etw run 5.

Fig. 3 shows the number of moves performed by the existing heuristics under ADHS-AILLA for two instances. ADHS behaves differently during the run by selecting certain heuristics more often and excluding others relying on the

performance changes. In addition, its behaviour is different on different instances. In conclusion, high-level search approaches utilising a group of low-level algorithms require online learning for better judgement of the changing characteristics of the search space.

In general, the quality of the solutions for instances with extended time windows is better than for instances with tight time windows. This is a useful result to show the effectiveness of softening time windows, in accordance with Taillard et al. 1997.

6. Conclusion

The presented routing and rostering problem contains the characteristics of two combinatorial optimization problems. In the literature, it is common to encounter problems with only a routing aspect, e.g. the vehicle routing problem, or a personnel rostering aspect, e.g. the nurse rostering problem. In this paper, a problem was modeled that combines both routing and rostering characteristics concerning the assignment of jobs to a security crew. A suite of generic approaches, i.e. hyper-heuristics, was introduced for addressing a set of varying instances. A detailed performance analysis was conducted based on experiments with real-world data. The results show that the generic hyper-heuristics can generate satisfactory results, i.e. schedules which could be used in practice.

In the future, the problem will be extended by increasing the number of visit types, adding resource availabilities and providing multiple contract options. For the hyper-heuristics, a feedback mechanism will be built allowing communication between hyper-heuristic components.

Acknowledgement This research was conducted in the context of a project funded by IWT (Flemish Institute for Science and Technology).

References

1. Burke, E. K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H.: The State of the Art of Nurse Rostering. J. Sched. 7(6), 441--449 (2004)
2. Cordeau, J.-F., Laporte, G., Mercier, A.: A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows. JORS 52, 928--936 (2001)
3. Polacek, M., Hartl, R. F., Doerner, K., Reimann, M.: A Variable Neighborhood Search for the Multi Depot Vehicle Routing Problem with Time Windows. J. Heur. 10(6), 613--627 (2004)
4. Ostertag, A., Doerner, K. F., Hartl, R. F., Taillard, E. D., Waelti, P.: POPMUSIC for a real-world large-scale vehicle routing problem with time windows. JORS 60**,** 934--943 (2009)
5. Burke, E.K., Kendall, G., Soubeiga, E., A Tabu-Search Hyperheuristic for Timetabling and Rostering. J. Heur. 6(9), 451--470 (2003)
6. Bilgin, B., De Causmaecker, P., Vanden Berghe, G.: A Hyperheuristic Approach to Belgian Nurse Rostering Problems. In: 4th Multidisciplinary International Scheduling Conference: Theory and Applications, pp. 683--689 (2009)
7. Bertels, S., Fahle, T.: A hybrid setup for a hybrid scenario: combining heuristics for the home care problem. Comp. and Oper. Res. 33(10), 2866--2890 (2006)

8. Begur, S.V., Miller, D.M., Weaver, J.R., An integrated spatial DSS for scheduling and routing home-health care nurses. Interfaces 27(4), 35--48 (1997)
9. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R.: A classification of hyper-heuristic approaches. Handbook of Metaheuristics, 449--468 (2010)
10. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: 3rd International Conference on Practice and Theory of Automated Timetabling, pp. 176--190 (2001)
11. Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. In: Metaheuristics: Computer Decision-Making, 523--544 (2003)
12. Ozcan, E., Misir, M., Ochoa, G., Burke E.K.: A Reinforcement Learning - Great-Deluge Hyper-Heuristic for Examination Timetabling. IJAMC 1(1): 39--59 (2010)
13. Bai, B., Kendall, G., An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-heuristics. Meta-heuristics: Progress as Real Problem Solvers, Selected Papers from the 5th Metaheuristics International Conference, 87--108 (2005)
14. Burke, E.K., Kendall, G., Misir, M., Ozcan, E.: Monte Carlo hyper-heuristics for examination timetabling. Ann. Oper. Res., 1--18 (2010)
15. Han, L., Kendall, G.: An Investigation of a Tabu Assisted Hyper-Heuristic Genetic Algorithm. In: the IEEE Congress on Evolutionary Computation, 3, pp. 2230--2237 (2003)
16. Burke, E.K., MacCarthy, B., Petrovic, S., Qu, R.: Knowledge Discovery in a Hyperheuristic Using Case-Based Reasoning on Course Timetabling. In: the 4th International Conference on the Practice and Theory of Automated Timetabling, pp. 276--286 (2002)
17. Burke, E.K., Hyde, M.R., Kendall, G., Evolving Bin Packing Heuristics with Genetic Programming. In: 9th Parallel Problem Solving from Nature, LNCS 4193, 860--869 (2006)
18. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R., A Survey of Hyper heuristics. CS Tech. Rep. University of Nottingham (2009)
19. Bilgin, B., Demeester, P., Misir, M., Vancroonenburg, V., Vanden Berghe, G., One hyper-heuristic approach to two timetabling problems in health care. Technical Report KAHO Sint-Lieven (2010)
20. Misir, M., Wauters, T., Verbeeck, K., Vanden Berghe, G., A new learning hyper-heuristic for the traveling tournament problem. In: 8th Metaheuristic International Conference (2009)
21. Taillard, E., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y, A tabu search heuristic for the vehicle routing problem with soft time windows. Trans. Sci. 31, 170–186 (1997)