An Investigation of Heuristic Decomposition to Tackle Workforce Scheduling and Routing With Time-dependent Activities Constraints

Wasakorn Laesanklang, Dario Landa-Silva, and J. Arturo Castillo-Salazar

School of Computer Science, ASAP Research Group, The University of Nottingham
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, United Kingdom.
{psxwl3, dario.landasilva}@nottingham.ac.uk, jacastillo.salazar@gmail.com

Abstract. This paper presents an investigation into the application of heuristic decomposition and mixed-integer programming to tackle workforce scheduling and routing problems (WSRP) that involve time-dependent activities constraints. These constraints refer to time-wise dependencies between activities. The decomposition method investigated here is called repeated decomposition with conflict repair (RDCR) and it consists of repeatedly applying a phase of problem decomposition and sub-problem solving, followed by a phase dedicated to conflict repair. In order to deal with the time-dependent activities constraints, the problem decomposition puts all activities associated to the same location and their dependent activities in the same sub-problem. This is to guarantee the satisfaction of time-dependent activities constraints as each sub-problem is solved exactly with an exact solver. Once the assignments are made, the time windows of dependent activities are fixed even if those activities are subject to the repair phase. The paper presents an experimental study to assess the performance of the decomposition method when compared to a tailored greedy heuristic. Results show that the proposed RDCR is an effective approach to harness the power of mixed integer programming solvers to tackle the difficult and highly constrained WSRP in practical computational time. Also, an analysis is conducted in order to understand how the performance of the different solution methods (the decomposition, the tailored heuristic and the MIP solver) is affected by the size of the problem instances and other features of the problem. The paper concludes by making some recommendations on the type of method that could be more suitable for different problem sizes.

Keywords: workforce scheduling and routing problem, time-dependent activities constraints, mixed integer programming, problem decomposition

1 INTRODUCTION

This paper applies Repeated Decomposition with Conflict Repair (RDCR) on a mixed integer programming model to tackle a Workforce Scheduling and Routing Problem (WSRP) with time-dependent activities constraints. The WSRP refers to assigning employees with diverse skills to a series of visits at different locations. A visit requires certain skills so that the tasks or activities can be performed. The problem has become important especially in the recent years because the number of businesses using a mobile workforce is growing [9]. These businesses usually provide services to people at their home. Examples are home care [2, 7], home healthcare [1], security patrol services [15, 19], technician scheduling [10, 14, 16], etc. In this type of scenarios, a mobile workforce must travel from its base to visit multiple locations to deliver services.

Time-dependent activities constraints refer to the case in which visits are time-wise related, such feature in WSRP was discussed in [4]. There are five types of time-dependent activities constraints: synchronisation, overlap, minimum difference, maximum difference and minimum-maximum difference. There have been attempts to use mathematical programming to find optimal solutions for WSRP [20, 4]. The problems are usually formulated as mixed integer programs (MIP) and implemented as a network flow problem. However, solving the problem using mathematical programming solvers requires very high computational time. Such solvers are able to find optimal solutions for only small instances and only sometimes feasible solution can be found within 4 hours. It has also been shown that when solving larger instances (e.g. more than 150 visits), it is often not possible to find optimal solutions due to the computer memory being exhausted. A constructive greedy heuristic (GHI) was proposed to solve WSRP with time-dependent activities constraints [3]. That algorithm provided better solutions than the mathematical programming solver when the number of visits is more than 100. In addition, other solution methods such variable neighbourhood search [17] and greedy constructive heuristics [23] have also been applied to WSRP instances with time-dependent activities constraints.

In the literature, there are works applying mathematical programming solvers within a decomposition approach to solve real-world problems. A decomposition method breaks a problem into smaller parts which are easier to solve. A problem can be decomposed by exact or heuristic approaches. An example of exact decomposition is Dantzig-Wolfe decomposition where all possible assignment combinations can be generated [6]. This approach may require high computational times to achieve a good solution. Heuristic decomposition generates sub-problems by splitting the full problem using some heuristic procedure to solve each sub-problem and integrate the partial solutions into a solution to the full problem. This usually means that heuristic decomposition does not guarantee optimality in the overall solution. An example of heuristic decomposition method is the Geographical Decomposition with Conflict Avoidance (GDCA) proposed in [11, 13] to tackle a home healthcare scheduling problem. Those home healthcare instances tackled with GDCA had a fixed time for the visits instead of a time window and had no time-dependent activities constraints. The GDCA technique decomposed a problem by geographical regions resulting in several sub-problems which then are tackled individually. The GDCA method was capable of finding a feasible solution even for instances with more than 1,700 clients. Other related heuristic decomposition methods using some form of clustering have been presented in the literature. For example, a large vehicle routing problem was decomposed into various clusters of customers assigned to a vehicle in [21].

This paper applies a Repeated Decomposition with Conflict Repair (RDCR) approach to a varied set of WSRP instances that involve time-dependent activities constraints. In general, RDCR decomposes a problem into sub-problems which then are individually solved with a mathematical programming solver. A sub-problem solution gives a path or sequence of visits for each employee. However, since an employee may be used in several sub-problems, this can lead to having *conflicting paths*, i.e. different paths that are assigned to the same employee. Another type of conflict are *conflicting assignments*, i.e. visits overlapping in time assigned to the same employee. Avoiding conflicting assignments within a path is guaranteed by the mathematical programming model. However, conflicting paths can arise because sub-problems are individually solved and the available workforce is shared among sub-problems. Therefore, conflicting paths need to be resolved by a conflict repair process described later in this paper. The stage of problem decomposition and sub-problem solving is followed by the stage of conflict repair. These two stages are repeatedly applied as part of the RDCR method until no more visits can be assigned in the current solution. This paper compares the solution quality from the proposed decomposition method to the results produced by the greedy constructive heuristic (GHI) presented in [3]. Moreover, this paper also conducts an in-depth analysis of the performance by the RDCR and GHI methods in respect of the problem features. This analysis aims to identify the types of problem instances in which each of the methods performs better. This will contribute to a better understanding of what type of approach is expected to be more successful according to the features of the problem instance in hand. It would also help to understand what problem features appear to present more difficulty for each method in order to identify directions for further research.

One contribution of this paper is a decomposition method that is adapted to tackle the WSRP with time-dependent activities constraints. The method represents a suitable approach to harness the power of mathematical programming solvers to tackle difficult instances of the WSRP. Another contribution of this paper is a better understanding of the performance by the decomposition method and the tailored constructive heuristic in respect of the problem features. The rest of the paper is organised as follows. Section 2 describes the workforce scheduling and routing problem. Section 3 presents the repeated decomposition with conflict repair method and it also introduces the modification for time-dependent activities constraints. Section 4 presents experimental results from comparing RDCR to the GHI greedy heuristic algorithm. Section 5 concludes the paper.

2 PROBLEM DESCRIPTION

This section describes the workforce scheduling and routing problem with time-dependent activities constraints. The MIP model to solve this problem was originally presented in [20] for a home care crew scheduling scenario. The model is also described in [3, 12] and hence not replicated here. As discussed in the introduction, time-dependent activities constraints arise from situations in which visits relate to each other time-wise. Hence, this section also describes the constraints of this type and their formulation.

2.1 Workforce Scheduling and Routing Problem

A network flow model was proposed in [20] for a home care crew scheduling scenario, which is an example of what is called here the workforce scheduling and routing problem (WSRP). That model balances the number of incoming edges and outgoing edges in each node corresponding to a visit location. An edge represents a worker arriving or leaving the visit location. Hence, such balancing means that a worker assigned to a visit must leave the location after performing the task and then move to the next visit location or to the depot. This balancing constraint is applied to each location visit except the depot which is considered as the source and the sink in the network flow model. This same model was also used in [3, 12] where other WSRP scenarios with time-dependent activities constraints were tackled using a greedy heuristic algorithm.

The model is a minimisation problem where the objective function is a summation of three main costs. First is the *deployment cost* of assigning each employee to visits. Second is the *preferences cost* of not assigning the most preferred employee to that visit. Third is the *unassigned visit cost* applied when a visit is left unassigned. Each of the three main costs in the objective function is multiplied by weights to give some level of priority to each cost. Here, the values for these weights are set as in [20].

The mixed-integer programming (MIP) model for the problem includes the following constraints. A visit is either assigned to employees or left unassigned. A visit can only be assigned to employees who are qualified to undertake activities associated to the visit. Each path must start from the employee's initial location and end at the final location. The flow conservation constraint guarantees that once employee k arrives to a visit location it then leaves that location in order to form a working path. Visits must start in their starting time window. Assignments of visits to employees must respect the employee's time availability. The time allocated for starting a visit must respect the travel time needed after completing the previous visit. The method presented in this paper has been adapted to tackle time-dependent activities constraints in particular. Such constraints indicate that time-wise dependencies exist between some visits and the specific constraints tackled here are described in more detail next.

2.2 Time-dependent Activities Constraints

A key difference with previous work described in [11] is that the WSRP scenarios tackled here include a special set of constraints called *time-dependent activities constraints* that establish some inter-dependence between activities. These constraints reduce the flexibility in the assignment of visits to employees because for example, a pair of visits might need to be executed in a given order. There are five constraint types: *overlapping*, *synchronisation*, *minimum difference*, *maximum difference* and *minimum-maximum difference*. A solution that does not comply with the satisfaction of these *time-dependent activities constraints* is considered infeasible.

- *Overlapping* constraint means that the duration of one visit i must extend (partially or entirely) over the duration of another visit j. This constraint is satisfied if the end time of visit i is later than the start time of visit j and also the end time of visit j is later than the start time of visit i.
- *Synchronisation* constraint means that two visits must start at the same time. This constraint is satisfied when the start times of visits i and j are the same.
- *Minimum difference* constraint means that there should be a minimum time between the start time of two visits. This constraint is satisfied when visit j starts at least for a given time units after the start time of visit i.
- *Maximum difference* constraint means that there should be a maximum time between the start time of two visits. This constraint is satisfied when visit j starts at most a given time units after the start time of visit i.
- *Minimum-maximum difference* constraint is a combination of the two previous conditions and it is satisfied when visit j starts at least a time units but not later than another time units after the start time of visit i.

3 THE DECOMPOSITION METHOD

This section describes the Repeated Decomposition with Conflict Repair (RDCR) approach used to tackle the WSRP with time-dependent activities constraints. A previous paper [11] presented a method called Geographical Decomposition with Conflict Avoidance (GDCA). In that work, conflicting paths and conflicting assignments as described above were not allowed to happen. However, the existence of time-dependent activities constraints makes it more difficult to just avoid such conflicts when assigning employees to visits. The RDCR method proposed here again seeks to harness the power of exact optimisation solvers by repeatedly decomposing and solving the given problem while also repairing the conflicting paths and conflicting assignments that may arise. The overall RDCR method is presented in Algorithm 1 and outlined next.

The RDCR method takes a WSRP problem denoted by $P = (K, C)$, where C is a set of visits and K is a set of available employees, and applies two main stages. One stage is *problem decomposition* and *sub-problem solving* (lines 2 to 5). The other stage is *conflict repair stage* (lines 6 to 10). The output of RDCR is a solution made by a set of valid paths, each of which is an ordered list of visits assigned to an employee. A valid path is assigned to exactly one employee and does not violate any of the constraints defined in Section 2. However, the problem decomposition and sub-problem solving stage may produce conflicting paths, i.e. two or more paths assigned to the same employee. These conflicting paths are then tackled by the conflict repair stage and converted into valid paths. Some

Algorithm 1: Repeated Decomposition and Conflict Repair

```
   Data: Problem $P = (K, C)$ where $K$ is a set of available workforce and $C$ is a
         set of unassigned visits
   Result: {SolutionPaths} FinalSolution
1  repeat
2      {Problem} $S$ = ProblemDecomp($K$, $C$);
3      for $s \in S$ do
4          sub_sol($s$) = cplex.solve($s$);
5      end
6      {Problem} $Q$ = ConflictDetection(sub_sol);
7      FinalSolution.add(NonConflict(sub_sol));
8      for $q \in Q$ do
9          cRepair_sol($q$) = cplex.solve($q$);
10     end
11     FinalSolution.add(cRepair_sol);
12     Update_UnassignedVisits($C$);
13     Update_AvailableWorkforce($K$);
14 until No assignment made;
```

visits that were already assigned in the conflicting paths might become unassigned as a result of the repairing process. These unassigned visits are then tackled by repeating the stages of problem decomposition and sub-problem solving followed by conflict repair over some iterations until no more visits can be assigned. The following subsections describe the RDCR method in more detail.

3.1 Problem Decomposition

The problem decomposition (line 2 in Algorithm 1) aims to reduce the size of the feasible region and hence makes possible to tackle the problem with an MIP solver. This process splits the problem into several sub-problems. Each of these sub-problems is made of a subset of employees and visits from the full-size problem but still considering all the types of constraints as in the model described in Section 2. Let S be a set of sub-problems $s = (K_s, C_s) \in S$ where K_s and C_s are the subsets of employees and visits respectively for sub-problem s. The outline of the problem decomposition process is shown in Algorithm 2. The two main steps are the visit partition (line 1) and the workforce selection (line 3). These two processes are described in detail next.

Visit Partition Algorithm 3 shows the steps for the visit partition process. It takes the set of visits C in a full-size problem and produces a partition S consisting of subsets of visits C_i. First, the set of visits C is grouped by location into *visitsList* (since two or more visits might be associated to the same geographical location). Then, each visit c in *visitsList* is allocated to a subset C_i. Basically, the algorithm puts visits that share the same location and visits that are time-dependent into the same subset. The aim of this is that when solving

Algorithm 2: Problem Decomposition

```
Data: {Workforce} $K$, {Visits} $C$
Result: {Problem} S is a collection of decomposition sub-problems.
1 $VP$ = VisitPartition($C$);
2 for $C_i \in VP$ do
3     $ws$ = WorkforceSelection($K$,$C_i$);
4     $S$.add(subproblem_builder($C_i$,$ws$));
5 end
```

each sub-problem, it becomes easier to enforce the time-dependent activities constraints.

Also, the algorithm observes a maximum size for each subset C_i or sub-problem. This is to have some control over the computational difficulty of solving each sub-problem. As it would be expected, the larger the sub-problem the more computational time required to find an optimal solution or even a feasible one with the MIP solver. However, partitioning into too small sub-problems usually results into solutions of low quality overall. Hence, the sub-problem size is set at 12 visits in our method. However, it is possible for a sub-problem to have more than 12 visits if this means having all activities with the same location and the corresponding time-dependent activities, grouped in the same sub-problem (see line 5 of Algorithm 3).

Workforce Selection Algorithm 4 shows the steps for the workforce selection process. It takes a subset of visits C_i and the set of employees K to then select a subset of employees ws for the given sub-problem. Basically, for each visit c in C_i the algorithm selects the lowest cost employee w from those employees who are not already allocated to another visit in this same sub-problem (see line 2 of Algorithm 4). That is, an employee w selected for visit c will not be available for another visit in C_i. This process gets a set of employees no larger than $|C_i|$. Note that this method does not generate a partition of the workforce K. This is because although a employee w may be selected for only one visit within subset C_i, such employee w could still be selected for another visit in a different sub-problem, hence potentially generating conflicting paths.

3.2 Sub-problem Solving

The problem decomposition process produces a set of sub-problems each with a subset of activities and a subset of selected employees. Each sub-problem is still defined by the MIP model presented in Section 2 with its corresponding cost matrix and other relevant parameters. Then, each sub-problem is tackled with the MIP solver (line 4 in Algorithm 1). Solving a sub-problem returns a set of paths. Once the sub-problems are solved there might be conflicting paths, i.e. paths in different sub-problems assigned to the same employee. The conflicting paths require additional steps to resolve the conflict while the valid paths can

Algorithm 3: Visit Partition Module

```
Data: {Visits} $C$
Result: {{Visits}} $VP = \{C_i | i = 1, \ldots, |S|\}$; Partition set of visits
visitsList = OrderByLocation($C$);
i = 0;
for $c \in visitsList$ do
    for $j$ = 1,...,$i$ do
        if $|C_j| <$ subproblemSize or c.shareLocation($C_j$) then
            $C_j$.add($c$);
            if c.hasTimeDependent then
                Visit $c_2$ = PairedVisit($c$);
                $C_j$.add($c_2$);
            end
        end
    end
    if c.isNotAllocated then
        $i$=$i$+1;
        $C_i$.add($c$);
        if c.hasTimeDependent then
            Visit $c_2$ = PairedVisit($c$);
            $C_i$.add($c_2$);
        end
    end
end
```

be used directly. The process to identify and repair such conflicting paths is explained next.

3.3 Conflict Repair

The conflict repair starts by identifying conflicting paths in the solutions to the sub-problems from the problem decomposition. All valid paths are immediately incorporated into the overall solution to the full-size problem. The process to detect conflicting paths is shown in Algorithm 5. It takes all sub-problems solutions and returns the set of conflicting paths Q. Basically, this process searches all sub-problem solutions and identifies all employees who are assigned

Algorithm 4: Workforce Selection Module

```
Data: {Visits} $C_i$, {Workforce} $K$
Result: {Workforce} $ws$
for $c \in C_i$ do
    Workforce $w$ = bestCostForVisit($K$,$c$,$ws$);
    $ws$.add($w$);
end
```

Algorithm 5: Conflict Path Detection Module

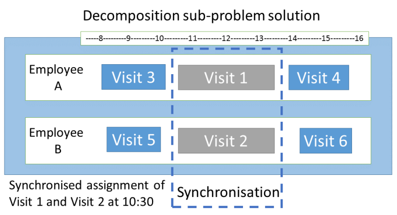
```
Data: {SolutionPaths} sub_sol; solutions from solving decomposition
      sub-problems
Result: {SolutionPaths} $Q$; Set of conflict paths
for {Path} $s_1 \in sub\_sol$ do
    for Path $a_1 \in s_1$ do
        SolutionPaths ConflictPath = null;
        pathConflicted=false;
        for $s_2 \in sub\_sol \,|s_2 \neq s_1$ do
            for Path $a_2 \in s_2$ do
                if $a_1.Employee = a_2.Employee$ then
                    ConflictPath.add($a_2$);
                    $s_2$.remove($a_2$);
                    pathConflicted = true;
                end
            end
        end
        if pathConflicted=true then
            ConflictPath.add($a_1$);
            $s_1$.remove($a_1$);
            $Q$.add(ConflictPath);
        end
    end
end
```

to two or more paths. It then groups those conflicting paths into sub-problems to repair. This sub-problem to repair has one employee and the set of activities from conflicting paths that belong to that employee.

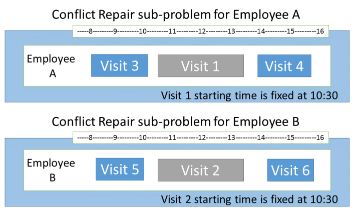
In order to repair conflicting paths, the MIP solver tackles the sub-problem to repair which results in a valid path and some unassigned visits. The valid path is incorporated to the solution of the full-size problem. The visits that remain unassigned are tackled by the next iteration of the problem decomposition and sub-problem solving stage followed by the conflict repair stage until no more assignments can be made.

3.4 Tackling Time-Dependent Activities Constraints

As described above, there are five types of time-dependent activities constraints: *overlapping*, *synchronisation*, *minimum difference*, *maximum difference* and *minimum-maximum difference*. Such time-dependent activities constraints are usually related to the assignment of two visits. Also, they usually require two employees, especially the synchronisation and overlapping cases. Hence, these constraints cannot be enforced by the conflict repair directly because the method builds a sub-problem to repair based on only one employee. Therefore, modifica-



(a) Solution to a decomposition sub-problem



(b) Conflict Repair sub-problems

Fig. 1. Example of tackling time-dependency on synchronised assignments. Sub-figure (a) shows the solution from solving a decomposition sub-problem. Sub-figure (b) shows two conflict repair sub-problem solutions. The assigned times from the decomposition sub-problem solution on visits with time-dependent activities (Visit 1 and Visit 2) are carried on to the later stage of the process. The time windows of Visit 1 and Visit 2 are fixed to the same value when preparing conflict repair sub-problem. Fixed starting time is enforced on Visit 1 and Visit 2 until both of them are incorporated into the final solution or the iterative process is terminated.

tion of the sub-problem to repair is necessary. This is mainly to keep the layout of assignments when time-dependent conditions are met.

Recall that the problem decomposition and sub-problem solving stage involves solving sub-problems in which visits share the same location and also time-dependent visits are grouped in the same sub-problem. Then, as defined by the MIP model, the solution to a sub-problem satisfies all time-dependent activities constraints. In order to keep the layout of time-dependent activities, visits of sub-problems in the conflict repair process require a fixed assigned time for every time-dependent activity. The fixed time is applied to time window, i.e. the earliest starting time is equal to the latest starting time for every time-dependent activity. Once the fixed time restriction is enforced, it affects every iteration of the process.

Figure 1 shows an example of how the modification works on a synchronisation constraint. With reference to the figure, suppose that visit 1 and visit 2 must be synchronised. Because visit 1 and visit 2 are time-dependent, they are grouped into the same decomposition sub-problem. The decomposition sub-problem is solved which gives paths for employee A and employee B, as shown in sub-figure 1(a). From that sub-figure, visit 1 and visit 2 are assigned to employee A and employee B, respectively. Both visits have their starting time set at 10:30. Suppose that both paths of employee A and employee B need to be repaired. At this stage, the time-dependent modification is applied. It overrides the time window of both visits and sets them to 10:30. Here, there are two sub-problems to repair, presented in sub-figure 1(b). Recall that a sub-problem to repair is defined based on an employee who has conflicting paths. Both sub-problems apply the new time window values forcing the start time of visit 1 and visit 2 to 10:30. The new time window is enforced until both visits are assigned to the final solution or the iterative process is terminated.

In the same way, the modification explained above tackles the other types of time-dependent constraints. The time-dependent visits are grouped in the same sub-problem and the solution of this part satisfies time-dependent activities constraints in the decomposition step. The time-dependent modification also applies when the time-dependent visit needs to be repaired. The modification replaces the time window of the visit by a fixed time given by the decomposition step. Then, this modification ensures that a solution that has gone through the conflict repair will satisfy the time-dependent activities constraints.

4 EXPERIMENTS AND RESULTS

This section describes the experiments carried out to compare the proposed RDCR method to the greedy heuristic (GHI) in [3] and better understand the success of each method according to features of the problem instances.

4.1 WSRP Instances Set

The RDCR method was applied to the set of WSRP instances presented in [4, 3]. Those problem instances were generated by adapting several WSRP from the literature. The instances are categorised in four groups: Sec, Sol, HHC and Mov. The Sec group contains instances from a security guards patrolling scenario [18]. The Sol group are instances adapted from the Solomon dataset [22]. The HHC group are instances from a home health care scenario [20]. Finally, the Mov group originates from instances of the vehicle routing problem with time windows [5]. The total number of instances accumulated in these four groups is 374.

4.2 Overview of Greedy Heuristic GHI

A greedy constructive heuristic tailored for the WSRP with time-dependent activities constraints was proposed in [3]. The algorithm starts by sorting visits according to some criteria such as visit duration, maximum finish time, maximum start time, etc. Then, it selects the first unassigned visit in the list and applies an *assignment process*. For each visit c, the *assignment process* selects all candidate employees who can undertake visit c (considering required skills and availability). If the number of candidate employees is less than the number of employees required for visit c, this visit is left unassigned. If visit c is assigned, visits that are dependent on visit c are processed. These dependent visits c' jump ahead in the *assignment process* and are themselves processed in the same way (i.e. processing other visits dependent on c'). The GHI stops when the unallocated list is empty and then returns the solution.

4.3 Computational Results

The proposed RDCR method was applied to the 374 instances and the obtained solutions were compared to the results reported by the greedy heuristic (GHI).

Table 1. Statistical result from Related-Samples Wilcoxon Signed Rank Test provided by SPSS

Total N		374
# of (RDCR < GHI),	RDCR is better than RDCR	209
# of (RDCR > GHI),	GHI is better than GHI	165
Test Statistic		37,806
Standard Error		2,092
Standardized Test Statistic		1.311
Asymp. Sig. (2-sided test)		.190

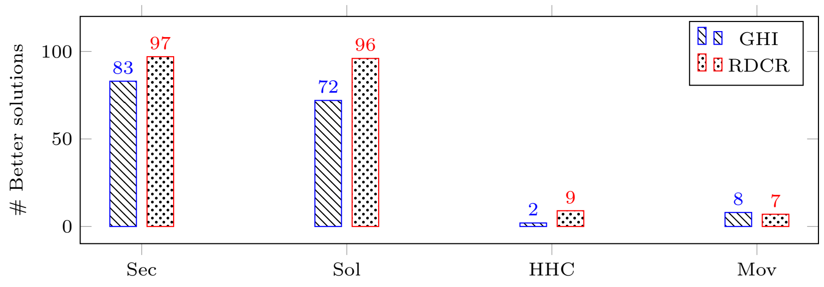


Fig. 2. Number of best solutions obtained by GHI and RDCR for each dataset.

First, the related-samples Wilcoxon Signed Rank Test [8] was applied to examine the differences between the two algorithms, GHI and RDCR. The significant level of the statistical test was set at $\alpha = 0.05$. Results of this statistical test using SPSS are shown in Table 1 showing that RDCR produced better solutions for 209 out of the 374 instances. However, there was no statistical significant difference on the solution quality between the two methods.

Figures 2 and 3 compare the number of best solutions found by each of the two methods and the average relative gap to the best known solutions. In these figures, results are grouped by dataset. Note that the relative gap is calculated by $\Delta = |z - z^b|/|z^b|$ where z represents an objective value of a solution and z^b

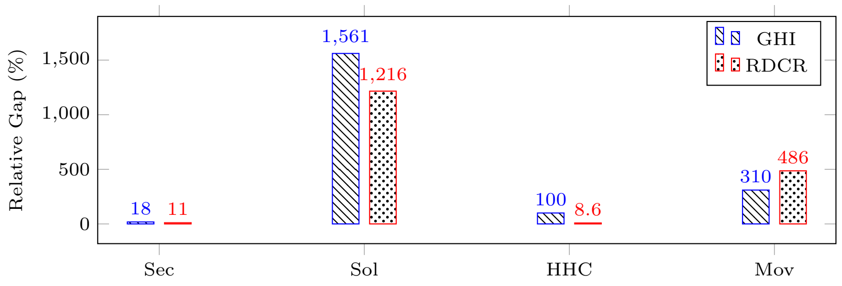


Fig. 3. Average relative gap (relative to the best known solution) obtained by GHI and RDCR. The lower the bar the better, i.e. the closer to the average best known solution.

is an objective value of the best known solution. Regarding the number of best solutions, RDCR produced better results than GHI on three datasets: Sec, Sol and HHC. Results also show that RDCR found lower values of average relative gap on the same three datasets. On the Mov dataset, GHI performed better in terms of number of best solutions and average relative gap.

On datasets Sec and Sol, RDCR found slightly better results than GHI as shown by the number of best solutions and the average relative gap. In dataset Sec, RDCR and GHI gave 11% and 18% of average relative gap respectively. This indicates that both algorithms provide good solution quality compared to the best known solution. On the other hand, both RDCR and GHI produced 1,216% and 1,561% respectively for the average relative gap to the best known solution in dataset Sol. This implies that both algorithms failed to find solutions that are of competitive quality to the best known solution, but both algorithms are competitive between them. It can be seen that instances in this Sol dataset are particularly difficult as neither the GHI heuristic nor the RDCR decomposition technique could produce solutions of similar quality to the best known solution.

On dataset HHC, the average relative gap of RDCR is much lower than the average gap of GHI. The results show that RDCR has 8.6% relative gap while GHI has 100%. For the HHC instances, RDCR found the best known solution for 9 instances and GHI found the best known solution for the other 2 instances. For these two instances, average relative gap of RDCR is 47%. However, in the 9 best solutions of RDCR, average gap of GHI is 109%. A closer look at the Sol dataset showed that these instances have priority levels defined for the visits. It turns out that GHI does not have sorting parameters to support such priority for visits because the algorithms sorting parameters focus on the time and duration of visits. On the other hand, RDCR implemented priority for visits within the MIP model. This could be the reason that explains the better results obtained by RDCR on this particular dataset.

On dataset Mov, GHI gives better performance. GHI delivers 8 better solutions (7 best known) from 15 instances while RDCR gives 7 better solutions (4 best known). The average relative gap of GHI is 310% which is less than the 486% relative gap provided by RDCR. There are 5 instances which best known solution is given by the mathematical programming solver. For these, the average relative gaps to the best known given by GHI is 315% and by RDCR is 36% respectively. It was found that the decomposition method does not show good performance on this particular Mov dataset, especially on instances with more than 150 visits. The main reason is that the solver cannot find optimal solutions to the sub-problems within the given time limit. Therefore, the size of sub-problems in these Mov instances should be decreased to allow for the sub-problems to be solved to optimality.

Figure 4 shows the cumulative distribution of RDCR and GHI solutions over the relative gap. It shows the number of solutions which have a relative gap to the best known less than the corresponding value in the X-axis. Note that 0% relative gap refers to the best known solution. For this case, GHI provides 115 best known solutions which is better than RDCR which provides 84 best solutions. This is

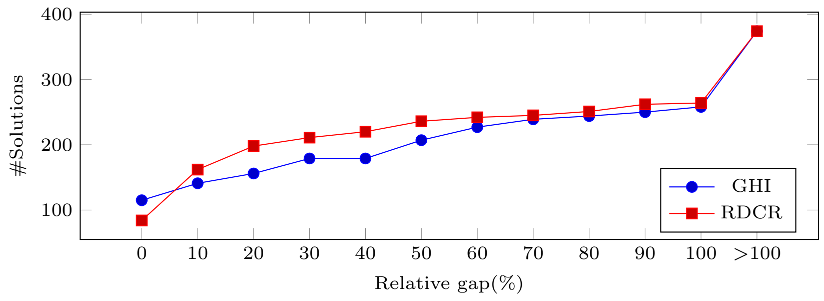


Fig. 4. Cumulative distribution of GHI and RDCR solution over the relative gap.

represented by the two leftmost points in the figure. However, from the value of 10% relative gap onwards, RDCR delivers larger number of solutions than GHI. Overall, apart from the overall number of best known solutions, RDCR provides higher number (or equal) of solutions than GHI for different values of relative gap. For example, if the solution acceptance rate is set at 50% relative gap, RDCR produces 236 solutions of this quality while GHI produces 207. Hence, RDCR delivers overall more solutions with acceptance rate up to 100% gap to the best known.

Figure 5 shows the distribution of computational time spent by the proposed RDCR method when solving the WSRP instances considered here. These results show that RDCR spends more computational time on most of the HHC instances with an overall average time spent on each instance of 2.4 minutes. Note that the highest computational time observed in these experiments is less than 74 minutes. On the other hand, the computational time spent by GHI is much shorter, taking less than one second on each instance. Therefore, GHI is clearly superior to RDCR in terms of computational time.

4.4 Performance According to Problem Difficulty

This part seeks to better understand the performance of the two algorithms GHI and RDCR. For this, a more detailed analysis is conducted of the instances in which each of the algorithms performs better than the other one. Then, the problem features are analysed in detail in order to unveil any conditions under which each of the algorithms appears to performs particularly well.

Table 2 presents the main characteristics of the problem instances in three groups. *Set All* has the 374 instances. *Set GHI* has all problem instances in which GHI produced better solutions than RDCR. *Set RDCR* has all problem instances in which RDCR produced better solutions than GHI. The table shows average and standard deviation values for 8 problem characteristics: the number of employees (#Emp), the number of visits (#Visit), visit duration (VisitDur), the number of time-dependent activities (#TimeDep), employee-visit ratio (Emp/Visit), employee available hours (EmpHours), average visit time window (VisitWindow),

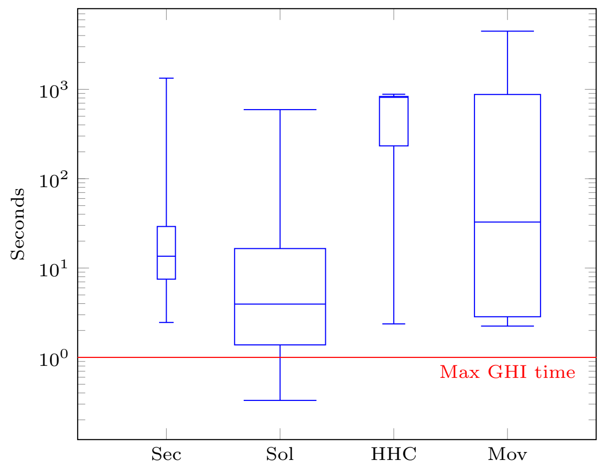


Fig. 5. Box and Whisker plots showing the distribution of computational time in seconds spent by RCDR for each group of instances. The wider the box the larger the number of instances in the group. The orange straight line presents the upper limit in the computational time spent by GHI (fixed to 1 second). The Y-axis is in logarithmic scale.

and planning horizon (Horizon). These values are presented in the table in the format Mean ± SD. Those problem characteristics for which there is a statistical significant difference between *Set GHI* and *Set RDCR* (using t-test at significance level $\alpha = .05$) are marked with *.

It seems obvious to relate the difficulty of a particular problem instance to it size, which can be measured by the number of employees and the number of visits. It could also be assumed that the length of the planning horizon might have some influence on the difficulty of the problem in hand, although perhaps to a lesser extent than the number of employees and visits. However, the analysis presented here seeks to identify other problem characteristics that might have an effect of the difficulty of the instances when tackled by each of the algorithms RDCR and GHI. For example, it can be argued that having visits with longer duration or large number of time-dependent activities could make the problem instance more difficult to solve because of the higher likelihood of time conflicts arising. In contrast, the difficulty could decrease for a problem instance that has higher employee to visit ratio (i.e. more workers to choose from), longer employee working hours or wider visit time windows (i.e. more flexibility for the assignment of visits).

Considering the above, it seems from Table 2 that instances in *Set RDCR* are less difficult than those in *Set GHI*. In respect of the problem size, instances in *Set RDCR* are on average smaller than those in *Set GHI*, on the number of employees (#Emp) and also the number of visits (#Visit). In addition, instances in *Set RDCR* have shorter visit duration (VisitDur) and lower number of time-dependent activities (#TimeDep) than instances in *Set GHI*. Moreover, note that although

Table 2. Summary of the problem features for different groups of problem instances. The *Set All* includes all instances. The *Set GHI* includes the instances in which GHI produces better solutions than RDCR. The *Set RDCR* includes the instances in which RDCR produces better solutions than GHI. Values are displayed in the format mean ± std. dev.

	Set All	*Set GHI*	*Set RDCR*
# Instances in Group	374	165	209
Problem Size			
#Emp*	22.5 ± 22.55	31.37 ± 27.86	15.49 ± 16.45
#Visit*	87.22 ± 53.23	117.6 ± 54.65	63.26 ± 37.83
Characteristics on Visits and Employees			
VisitDur*	214.3 ± 198.8	254.6 ± 221.6	182.4 ± 173.4
#TimeDep*	13.95 ± 10.03	18.87 ± 10.63	10.37 ± 7.91
Emp/Visit*	1.164 ± 0.072	1.156 ± 0.074	1.172 ± 0.079
EmpHours	20.8 ± 11.93	21.66 ± 11.45	20.11 ± 12.31
VisitWindow	392.9 ± 297.6	406.9 ± 325.9	381.9 ± 274.2
Horizon	1248 ± 715.9	1300 ± 687.2	1207 ± 738.4

* indicates statistical significant difference using t-test at significance level $\alpha = .05$.

the averages values of employee-visit ratio (Emp/Visit) are very similar for sets *Set RDCR* and *Set GHI*, the difference is still statistically significant. The differences between the two sets in respect of the remaining three problem characteristics, employee available hours (EmpHours), visit time window (VisitWindow) and planning horizon (Horizon) were found to be not statistically significant.

Then, from the above analysis it can be argued that the RDCR approach performs better than GHI on instances of lower difficulty level. However, establishing the boundary between lower and higher difficulty is not so clear given the overlap in values for the 8 problem characteristics between *Set RDCR* and *Set GHI*. Hence, the proposal here is to recommend the use of RDCR for instances with less than 22.5 employees and less than 87 visits (the average values considering all 374 instances), and the use of GHI otherwise. This recommendation can be used as a first step for choosing between RDCR and GHI.

4.5 Performance on Producing Acceptable Solutions

The previous subsection sought to identify a boundary in problem difficulty between those instances in which each of the methods RDCR and GHI performs better than the other one. This subsection seeks to identify instances for which both algorithms can deliver acceptable solutions. For this, a solution that has a relative gap of at most 100% with respect to the best known solution is considered acceptable, otherwise it is labelled unacceptable.

The first part of the analysis splits the problem instances into two groups. The group *Accept Heur* has instances for which an acceptable solution was found

Table 3. Summary of the problem features for different groups of problem instances. The group *Accept Heur* includes instances for which an acceptable solution was found by at least one of the two heuristic algorithms RDCR and GHI. The group *Reject Heur* includes instances for which none of RDCR or GHI delivers an acceptable solution. Values are displayed in the format mean ± std. dev.

# Instances in Group	*Reject Heur* 79	*Accept Heur* 295
Problem Size		
#Emp*	9.911 ± 4.249	25.87 ± 25.40
#Visit*	49.39 ± 21.17	97.34 ± 54.75
Characteristics on Visits and Employees		
VisitDur*	43.91 ± 53.54	259.89 ± 199.09
#TimeDep*	6.32 ± 3.11	15.99 ± 10.27
Emp/Visit	1.168 ± 0.048	1.164 ± 0.078
EmpHour*	18.02 ± 13.69	21.54 ± 11.34
VisitWindow	345.60 ± 387.58	405.58 ± 268.43
Horizon*	1081.41 ± 821.62	1292.61 ± 608.71

* indicates statistical significant difference using t-test at significance level $\alpha = .05$.

by at least one of the two heuristic algorithms RDCR and GHI. The group *Reject Heur* has instances for which none of RDCR or GHI delivers an acceptable solution. Basically, this analysis seeks to identify a boundary in problem difficulty for which the methods RDCR and GHI can perform better than an exact solver. Table 3 shows the problem characteristics for the two groups *Accept Heur* and *Reject Heur*. As before, each row shows the average and standard deviation values for each of 8 problem characteristics. Those problems characteristics for which there is a statistical significant difference between the two groups (using t-test at significance level $\alpha = .05$) are marked with *.

The results in Table 3 show that there are significant differences between the groups *Accept Heur* and *Reject Heur* on six problem characteristics. That is, the group *Accept Heur* shows higher mean values than the group *Reject Heur* for the number of employees (#Emp), the number of visits (#Visit), visit duration (VisitDur), the number of time-dependent activities (#TimeDep), employee available hours (EmpHours), and planning horizon (Horizon). These results indicate that GHI and RDCR do not provide acceptable solutions on the smaller instances with around 10 employees and 50 visits. However, these algorithms do well on the larger instances with around 26 employees and 97 visits. This is because the exact solver performs very well on the smaller instances but not so well when the problem size grows. Hence, the proposal here is to recommend the use of the exact solver for problems with less than 15 employees and 70 visits. For larger problem instances the solver may spend too long time finding solutions hence it is better to use GHI or RDCR considering the recommendation in the previous subsection.

Table 4. Summary of the problem features for different groups of problem instances. The group *Accept GHI* includes instances for which an acceptable solution was found by algorithm GHI, otherwise the instance is included in group *Reject GHI*. Values are displayed in the format mean ± std. dev.

	Reject GHI	*Accept GHI*
# Instances in Group	37	258
Problem Size		
#Emp*	15.97 ± 31.23	27.29 ± 24.19
#Visit*	54.86 ± 49.79	103.43 ± 52.77
Characteristics on Visits and Employees		
VisitDur*	42.06 ± 53.77	291.13 ± 192.69
#TimeDep*	8.00 ± 10.80	17.14 ± 9.69
Emp/Visit	1.1471 ± 0.07873	1.166 ± 0.0779
EmpHour	23.19 ± 20.27	21.30 ± 9.44
VisitWindow	462.90 ± 468.99	397.36 ± 226.01
Horizon	1391.78 ± 1216.20	1278.38 ± 566.81

[*] indicates statistical significant difference using t-test at significance level $\alpha = .05$.

The second part of the analysis analysis splits the 295 problem instances from the group *Accept Heur* into groups according to whether the particular method GHI or RDCR produces acceptable solutions or not. As before, a solution that has a relative gap of at most 100% with respect to the best known solution is considered acceptable, otherwise it is labelled unacceptable. Table 4 shows the split for method GHI into groups *Accept GHI* with 258 instances and *Reject GHI* with 37 instances. There are significant differences between the two groups on four characteristics: the number of employees (#Emp), the number of visits (#Visit), visit duration (VisitDur) and the number of time-dependent activities (#TimeDep) with larger values for the group *Accept GHI*. These results confirm that GHI provides acceptable solutions on the larger instances but it struggles to produce acceptable solutions for some smaller instances.

Table 5 shows the split for method RDCR into groups *Accept RDCR* with 264 instances and *Reject RDCR* with 31 instances. There are significant differences between the two groups on three characteristics: the number of employees (#Emp), visit duration (VisitDur) and employee-visit ratio (Emp/Visit). The size of instances in group *Accept RDCR* seems smaller than in group *Reject RDCR* as given by #Emp and #Visit, although only for #Emp the difference is significant. Instances in the group *Reject RDCR* have shorter visit duration and lower employee-visit ratio. A problem instance could become more difficult to solve if there are less workers to be assigned to visits. These results confirm that the performance of RDCR on providing acceptable solutions suffers as the size of the problem grows.

From the above analysis on producing acceptable solutions, some recommendations can be drawn in respect of what type of approach to use according to

Table 5. Summary of the problem features for different groups of problem instances. The group *Accept RDCR* includes instances for which an acceptable solution was found by algorithm RDCR, otherwise the instance is included in group *Reject RDCR*. Values are displayed in the format mean ± std. dev.

	Reject RDCR	*Accept RDCR*
# Instances in Group	31	264
Problem Size		
#Emp*	46.74 ± 54.91	23.42 ± 17.88
#Visit	155.6 ± 59.01	95.20 ± 53.95
Characteristics on Visits and Employees		
VisitDur*	41.19 ± 62.94	285.57 ± 193.80
#TimeDep	15.35 ± 7.38	16.07 ± 10.57
Emp/Visit*	1.077 ± 0.0654	1.174 ± 0.0731
EmpHour	20.82 ± 16.58	21.62 ± 10.60
VisitWindow	407.11 ± 453.92	405.40 ± 238.84
Horizon	1249.41 ± 995.01	1297.68 ± 636.26

[*] indicates statistical significant difference using t-test at significance level $\alpha = .05$.

Table 6. Type of approach recommended according to the problem size and number of instances in each size class.

Algorithm	Exact Method	RDCR	Heuristic	GHI
#Instance	79	37	227	31
Problem Size	Very Small	Small	Medium	Large
Average #Emp	9.91	15.97	23.42 - 27.29	49.74
Average #Visit	49.39	54.86	95.20 - 103.43	155.6

the problem size. Table 6 shows the type of approach recommended according to the problem size and number of instances in each size class. The first row of the table shows the suggested algorithm for each size class, Heuristic refers to either GHI or RDCR. For each size class, the table shows the number of instances (#Instance), the problem size label, the average number of employees (Average #Emp) and the average number of visits (Average #Visit). It is suggested that to use the exact method to solve very small instances, to use RDCR to solve small and medium instances and to use GHI to solve medium and large instances. The problem size class with the largest number of instances is the medium class for which the two heuristic algorithms, GHI and RDCR, find acceptable solutions. These recommendations in Table 6 were drawn from looking at the reject groups in Tables 3 to 5. Both GHI and RDCR do not perform well when solving small instances, given that group *Reject Heur* in Table 3 has the smallest average problem size. RDCR should be used for instances larger than those in group *Reject Heur*, Table 4 shows that the *Reject GHI* group has average problem size larger than the *Reject Heur* group and smaller than the *Reject RDCR* group.

GHI tends to be effective in the largest instance group, it can be seen from Table 5 that the *Reject RDCR* group has the largest average problem size compared to the *Reject GHI* group and *Reject Heur.* However, both RDCR and GHI have similar performance as their acceptable solutions are similar in number.

5 CONCLUSION

This paper presented a decomposition method for mixed integer programming to solve instances of the workforce scheduling and routing problem (WSRP) with time dependent activities constraints. The method uses heuristic partition and selection to split a problem into sub-problems. A sub-problem solution gives a path or sequence of visits for each employee. Each sub-problem is individually solved by the MIP solver. Within a sub-problem solution or path, all constraints are satisfied. Paths may conflict with paths from other sub-problems, i.e. two or more different sequences of visits but assigned to the same employee. This can be fixed by a conflict repair process. However, conflict repair requires modification to support time-dependent activities constraints since the repairing process may rearrange assignment time. Thus, the modification maintains the layout of time-dependent activities by fixing the assigned time of the time-dependent activities. Therefore, the solution from conflict repair does not violate any constraints.

The proposed RDCR approach is applied to solve four WSRP scenarios with a total of 374 instances. The experimental results showed that RDCR is able to find better solutions than the GHI heuristic for 209 out of the 374 instances. However, the statistical test showed that RDCR does not perform significantly different to the deterministic greedy heuristic (GHI). RDCR showed better performance on three out of four datasets. The computational time required to solve a problem instance with RDCR ranged from less than a second to 74 minutes. The average computational time was under 3 minutes. Overall, the proposed RDCR with time-dependent modification is able to effectively solve WSRP instances with time-dependent activities constraints. The method found competitive feasible solutions to every instance and within reasonable computational time.

The paper also conducts a study to investigate the performance of RDCR in respect of some problem features related to the problem size. The analysis has shown that RDCR provides better solutions particularly in smaller instances. Hence, instances with less than 87 visits and less than 22 workers should be tackled by RDCR to obtain higher quality solutions. Furthermore, another aim of the study was to determine the class of problem size that can be more effectively tackled with the heuristic approaches RDCR and GHI. For this, acceptable solutions are considered to be those that have a relative gap of no more than 100% with respect to the best known solution. The analysis revealed that RDCR and GHI work effectively in a wide range of problem sizes. The GHI method appears to be less effective on smaller instances while RDCR appears to be less effective on larger instances. Therefore, in order to produce acceptable solutions as defined here, it is recommended to use an exact solver for very small instances, to use RDCR for small and medium instances and to use GHI for medium and large instances.

As future work it is suggested to improve the computational time of the proposed RDCR approach. Such improvement might be achieved by applying different methods to partition the set of visits or by using more effective workforce selection rules. Also, determining the right sub-problem size could be interesting as it could help to balance solution quality and time spent on computation.

ACKNOWLEDGEMENTS

The authors are grateful for access to the University of Nottingham High Performance Computing Facility. Also, the first author thanks the DPST Thailand for partial financial support of this research.

References

1. Angelis, V.D.: Planning home assistance for AIDS patients in the City of Rome , Italy. Interfaces 28, 75–83 (1998)
2. Borsani, V., Andrea, M., Giacomo, B., Francesco, S.: A home care scheduling model for human resources. 2006 International Conference on Service Systems and Service Management pp. 449–454 (2006)
3. Castillo-Salazar, J.A., Landa-Silva, D., Qu, R.: A greedy heuristic for workforce scheduling and routing with time-dependent activities constraints. In: Proceedings of the 4th International Conference on Operations Research and Enterprise Systems (ICORES 2015) (2015)
4. Castillo-Salazar, J., Landa-Silva, D., Qu, R.: Workforce scheduling and routing problems: literature survey and computational study. ANN OPER RES (2014)
5. Castro-Gutierrez, J., Landa-Silva, D., Moreno, P.J.: Nature of real-world multi-objective vehicle routing with evolutionary algorithms. In: Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on. pp. 257–264 (2011)
6. Dantzig, G.B., Wolfe, P.: Decomposition principle for linear programs. OPER RES 8(1), 101–111 (1960)
7. Eveborn, P., Flisberg, P., Rönnqvist, M.: Laps Care–an operational system for staff planning of home care. EUR J OPER RES 171(3), 962–976 (2006)
8. Field, A.: Discovering Statistics Using IBM SPSS Statistics. SAGE Publication Ltd, London, UK, 4 edn. (2013)
9. Hiermann, G., Prandtstetter, M., Rendl, A., Puchinger, J., Raidl, G.R.: Metaheuristics for solving a multimodal home-healthcare scheduling problem. Central European Journal of Operations Research 23, 89–113 (2015)
10. Jean-François, C., Gilbert, L., Federico, P., Stefan, R.: Scheduling technicians and tasks in a telecommunications company. Journal of Scheduling 13(4), 393–409 (2010)
11. Laesanklang, W., Landa-Silva, D., Castillo-Salazar, J.A.: Mixed integer programming with decomposition to solve a workforce scheduling and routing problem. In: Proceedings of the 4th International Conference on Operations Research and Enterprise Systems (ICORES 2015). pp. 283–293 (2015)
12. Laesanklang, W., Landa-Silva, D., Castillo-Salazar, J.A.: Mixed integer programming with decomposition for workforce scheduling and routing with time-dependent activities constraints. In: Proceedings of 5th the International Conference on Operations Research and Enterprise Systems. pp. 330–339 (2016)

13. Laesanklang, W., Pinheiro, R.L., Algethami, H., Landa-Silva, D.: Extended decomposition for mixed integer programming to solve a workforce scheduling and routing problem. In: Operations Research and Enterprise Systems, Communications in Computer and Information Science, vol. 577, chap. 12, pp. 191–211. Springer International Publishing (2015)
14. Laugier, A., Anne-marie, B., Telecom, R.F.: Technicians and interventions scheduling for telecommunications. Francetelecom pp. 1–7 (2006)
15. Leigh, J., Jackson, L., Dunnett, S.: Police officer dynamic positioning for incident response and community presence. In: Proceedings of the 5th International Conference on Operations Research and Enterprise Systems (ICORES 2016). pp. 261–270 (2016)
16. Lesaint, D., Voudouris, C., Azarmi, N.: Dynamic workforce scheduling for British telecommunications plc. Interfaces 30, 45–56 (2000)
17. Mankowska, D., Meisel, F., Bierwirth, C.: The home health care routing and scheduling problem with interdependent services. Health Care Management Science 17(1), 15–30 (2014)
18. Misir, M., Smet, P., Verbeeck, K., Vanden Berghe, G.: Security personnel routing and rostering: a hyper-heuristic approach. In: Proceedings of the 3rd International Conference on Applied Operational Research, ICAOR2011. pp. 193–205 (2011)
19. Misir, M.and Smet, P.V.B.G.: An analysis of generalised heuristics for vehicle routing and personnel rostering problems. J Oper Res Soc 66(5), 858–870 (2015)
20. Rasmussen, M.S., Justesen, T., Dohn, A., Larsen, J.: The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. EUR J OPER RES 219(3), 598–610 (2012)
21. Reimann, M., Doerner, K., Hartl, R.F.: D-Ants: Savings based ants divide and conquer the vehicle routing problem. COMPUT & OPER RES 31(4), 563 – 591 (2004)
22. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problem with time window constraints. OPER RES 35(2) (1987)
23. Xu, J., Chiu, S.: Effective heuristic procedures for a field technician scheduling problem. J HEURISTICS 7(5), 495–509 (2001)