Mixed Integer Programming with Decomposition for 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, Nottingham, U.K.
{*psxwl3, dario.landasilva, psxjaca*}*@nottingham.ac.uk*

Keywords: Workforce Scheduling and Routing Problem, Time-dependent Activities Constraints, Mixed Integer Programming, Problem Decomposition.

Abstract: We present a mixed integer programming decomposition approach to tackle workforce scheduling and routing problems (WSRP) that involve time-dependent activities constraints. The proposed method 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. Five types of time-dependent activities constraints are considered: overlapping, synchronisation, minimum difference, maximum difference, and minimum-maximum difference. Experiments are conducted to compare the proposed method 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.

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. Generally, time-dependent activities constraints refer to the case in which visits are time-wise related, such feature in WSRP was discussed in (Castillo-Salazar et al., 2014).

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. Also, some visits may require more than one employee. The WSRP arises in different scenarios such as home health-care (Akjiratikarl et al., 2007), security personal routing (Misir et al., 2011), maintenance services scheduling (Cordeau et al., 2010), etc. A detailed study of the constraints and other problem features of the WSRP can be found in (Castillo-Salazar et al., 2014). That work showed that solving large instances of the WSRP to optimally is very challenging especially in cases with more than 100 visits. It also showed that finding optimal solutions with a mathematical solver either takes too long computational time or it is not possible due to the computer memory being exhausted. They found that a MIP solver was able to find feasible solutions within 2 hours of computation time. Later, a greedy heuristic algorithm was presented in (Castillo-Salazar et al., 2015) to tackle the WSRP with time-dependent activities constraints.

Basically, *time-dependent activities constraints* define a time-wise relation between two visits. There are five types of time-dependent activities constraints: synchronisation, overlap, minimum difference, maximum difference and minimum-maximum difference. Other solution methods such as mixed integer programming (Rasmussen et al., 2012), variable neighbourhood search (Mankowska et al., 2014) and other greedy heuristics (Xu and Chiu, 2001) have also been applied to WSRP instances with time-dependent activities constraints.

A mixed integer programming decomposition method, called Geographical Decomposition with Conflict Avoidance (GDCA), was proposed in (Laesanklang et al., 2015) to tackle a home care scheduling problem which is a type of WSRP. Those home care problem instances tackled with GDCA had a fixed time for the visits (instead of a time window) and no time-dependent activities constraints. The GDCA is considered a heuristic decomposition technique because it does not seek to reduce the gap between the lower and upper bounds. The technique decomposed a problem by geographical regions resulting in several sub-problems which then are tackled

Laesanklang, W., Landa-Silva, D. and Castillo-Salazar, J.
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 (ICORES 2016)*, pages 330-339
ISBN: 978-989-758-171-7

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, e.g. (Reimann et al., 2004) decomposed a large vehicle routing problem into various clusters of customers assigned to a vehicle.

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 leads 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 heuristic presented in (Castillo-Salazar et al., 2015).

The main 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. The rest of the paper is organised as follows. Section 2 presents the workforce scheduling and routing problem tackled here and the corresponding mixed integer programming (MIP) model. Section 3 describes 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 a greedy heuristic algorithm. Section 5 concludes the paper.

2 PROBLEM DESCRIPTION AND FORMULATION

This section describes the workforce scheduling and routing problem with time-dependent activities constraints and the mixed integer programming (MIP) model for this problem. The MIP model was originally presented in (Rasmussen et al., 2012) for a home care crew scheduling scenario. This scenario and several others are tackled here with the proposed solution technique. The type of WSRP tackled here is that involving time-dependent activities constraints, i.e. 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 MIP Model for WSRP

A network flow model was proposed by (Rasmussen et al., 2012) for a home care crew scheduling scenario, which is an example of what we call 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. The MIP model is given by equations (1) to (14) and Table 1 gives the notation used in the model separated into three parts: sets, parameters and variables. We note that this same model was also used in (Castillo-Salazar et al., 2015) where other WSRP scenarios with time-dependent activities constraints were tackled using a greedy heuristic algorithm.

The MIP model is a minimisation problem where the objective function (1) is a summation of three main costs. First is the *deployment cost* (denoted $c^k_{i,j}$) of assigning each employee k to visit i and then move to visit j (indicated by $x^k_{i,j}$). Second is the *preferences cost* (denoted δ^k_i) of assigning a lower preference employee to a visit, i.e. not assigning the most preferred employee to that visit. Third is the *unassigned visit cost* (denoted γ_i) applied when a visit is left unassigned (indicated by $y_i = 1$). Each of the three main costs in the objective function is multiplied by a weight (ω_1, ω_2 and ω_3 respectively) to give some level of priority to each cost. Here, the values for these weights are set as in (Rasmussen et al., 2012).

Table 1: MIP model notation for WSRP.

Sets	
K	A set of employees.
C	A set of visit locations.
N^k	A set of available locations for employee k defined by $N^k = C \cup \{0^k, n^k\}$.
$0^k, n^k$	Start and end locations respectively for employee k.
P	A set of time dependent of paired visits.
Parameters	
$[\alpha_i, \beta_i]$	Start time window of visit i.
$[A^k, B^k]$	Start and end working times respectively for employee n^k.
ρ_i^k	Binary parameter indicating that employee k is eligible for visit i based on skill requirement ($\rho_i^k = 1$) or not ($\rho_i^k = 0$).
$c_{i,j}^k$	Cost of deploying employee k to visit i and then move to visit j.
$s_{i,j}^k$	Duration of visit i and travel time from visit i to visit j for employee k.
$p_{i,j}$	Time-dependent parameter defined for a pair of visits i and j.
δ_i^k	Preference cost of deploy employee k to do visit i.
γ_i	Penalty cost of not assigning visit i.
$\omega_1, \omega_2, \omega_3$	Weights for the objective function.
Variables	
$x_{i,j}^k$	Binary decision variable with value 1 indicating that employee k is assigned to visit i and then move to visit j, and 0 otherwise.
y_i	Binary decision variable with value of 1 indicating that visit i is unassigned, and 0 otherwise.
t_i^k	Continuous variable for the time assigned to employee k starting visit i.

$$\text{Minimise} \quad \omega_1 \sum_{k\in K}\sum_{i\in N^k}\sum_{j\in N^k} c_{i,j}^k x_{i,j}^k + \omega_2 \sum_{k\in K}\sum_{i\in C}\sum_{j\in N^k} \delta_i^k x_{i,j}^k + \omega_3 \sum_{i\in C} \gamma_i y_i \quad (1)$$

Subject to

$$\sum_{k\in K}\sum_{j\in N^k} x_{i,j}^k + y_i = 1 \quad \forall i \in C \quad (2)$$

$$\sum_{j\in N^k} x_{i,j}^k \le \rho_i^k \quad \forall k \in K, \forall i \in C \quad (3)$$

$$\sum_{j\in N^k} x_{0^k,j}^k = 1 \quad \forall k \in K \quad (4)$$

$$\sum_{i\in N^k} x_{i,n^k}^k = 1 \quad \forall k \in K \quad (5)$$

$$\sum_{i\in N^k} x_{i,h}^k - \sum_{j\in N^k} x_{h,j}^k = 0 \quad \forall k \in K, \forall h \in C \quad (6)$$

$$\alpha_i \sum_{j\in N^k} x_{i,j}^k \le t_i^k \quad \forall k \in K, \forall i \in C \cup 0^k \quad (7)$$

$$t_i^k \le \beta_i \sum_{j\in N^k} x_{i,j}^k \quad \forall k \in K, \forall i \in C \cup 0^k \quad (8)$$

$$A^k \le t_i^k \le B^k \quad \forall k \in K, \forall i \in C \quad (9)$$

$$t_i^k + s_{i,j}^k x_{i,j}^k \le t_j^k + \beta_i (1 - x_{i,j}^k) \forall k \in K, \forall i, j \in N^k \quad (10)$$

$$\alpha_i y_i + \sum_{k\in K} t_i^k + p_{i,j} \le \sum_{k\in K} t_j^k + \beta_j y_j \quad \forall i, j \in P \quad (11)$$

$$x_{i,j}^k \quad \text{are binary,} \quad \forall k \in K, \forall i, j \in N^k \quad (12)$$

$$y_i \quad \text{are binary,} \quad \forall i \in C \quad (13)$$

$$t_i^k \ge 0 \quad \forall k \in K, \forall i \in N^k \quad (14)$$

The MIP model includes the following constraints. A visit is either assigned to employees or left unassigned (2). A visit can only be assigned to employees who are qualified to undertake activities associated to the visit (3). Each path must start from the employee's initial location (4) and end at the final location (5). The flow conservation constraint guar-

Table 2: Value of time-dependent parameter $p_{i,j}$ (constraint 11) for each of the five time-dependent activities constraints.

Constraint Types	$p_{i,j}$	$p_{j,i}$
Overlapping	$-d_j$	$-d_i$
Synchronisation	0	0
Minimum difference	p_i^l	N/A
Maximum difference	N/A	$-p_i^u$
Minimum-maximum difference	p_i^l	$-p_i^u$

Table 3: Conditions to validate the satisfaction of each time-dependent activities constraint.

Constraint Types	Validate Condition
Overlapping	$t_i^{k_1} + d_i > t_j^{k_2}$ $t_j^{k_2} + d_j > t_i^{k_1}$
Synchronisation	$t_i^{k_1} = t_j^{k_2}$
Minimum Difference	$t_i^{k_1} + p_i^l \le t_j^{k_2}$
Maximum Difference	$t_i^{k_1} + p_i^u \ge t_j^{k_2}$
Minimum-Maximum Difference	$t_i^{k_1} + p_i^l \le t_j^{k_2}$ $t_i^{k_1} + p_i^u \ge t_j^{k_2}$

- i, j is a pair of visits with some time dependency and both assigned in a solution.
- $t_i^{k_1}, t_j^{k_2}$ are the start times for visit i and j assigned to employees k_1 and k_2 respectively.
- d_i, d_j are the durations of visit i and visit j respectively.
- p_i^l, p_i^u are minimum difference and maximum difference duration respectively between visit i and visit j.

antees that once employee k arrives to a visit location it then leaves that location in order to form a working path (6). Visits must start in their starting time window as denoted by (7) and (8). Assignments of visits to employees must respect the employee's time availability (9). The time allocated for starting a visit must respect the travel time needed after completing the previous visit (10). The time-dependent activities constraints indicate that time-wise connections exist between some visits (11). The methodology presented in this paper has been adapted to tackle this type of time-dependent activities constraints in particular. Lastly, the decision variables in this MIP model are denoted by (12), (13) and (14).

2.2 Time-dependent Activities Constraints

A key difference with our previous work in (Laesanklang et al., 2015) 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 as denoted by (11). 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*. Table 2 shows the value given to the time-dependent parameter in constraint 11 for each type of time-dependent activity constraint. Table 3 presents the formulation for each of these constraints when $p_{i,j}$ has been applied. A solution that does not comply with the satisfaction of these *time-dependent activities constraints* as defined in Table 3 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. Therefore, $p_{i,j} = -d_j$ and $p_{j,i} = -d_i$.
- *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. Therefore, $p_{i,j} = p_{j,i} = 0$.
- *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 p_i^l time units after the start time of visit i. Therefore, $p_{i,j} = p_i^l$.
- *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 p_i^u time units after the start time of visit i. Therefore, $p_{j,i} = -p_i^u$.
- *Minimum-maximum difference* constraint is a combination of the two previous conditions and it is satisfied when visit j starts at least p_i^l time units but not later than p_i^u time units after the start time of visit i. Therefore, $p_{i,j} = p_i^l$ and $p_{j,i} = -p_i^u$.

3 REPEATED DECOMPOSITION WITH CONFLICT REPAIR

This section describes the Repeated Decomposition with Conflict Repair (RDCR) approach used to tackle the WSRP with time-dependent activities constraints. In a previous paper (Laesanklang et al., 2015) we 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 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.

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 ∈ 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 ∈ 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;
```

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.

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 ∈ VP do
3      ws = WorkforceSelection(K,C_i);
4      S.add(subproblem_builder(C_i,ws));
5  end
```

3.1.1 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 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, we set the sub-problem size 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).

3.1.2 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

Algorithm 3: Visit Partition Module.

```
   Data: {Visits} C
   Result: {{Visits}} VP = {C_i | i = 1,...,|S|};
           Partition set of visits
 1 visitsList = OrderByLocation(C);
 2 i = 0;
 3 for c ∈ visitsList do
 4     for j = 1,...,i do
 5         if |C_j| < subproblemSize or
            c.shareLocation(C_j) then
 6             C_j.add(c);
 7             if c.hasTimeDependent then
 8                 Visit c_2 = PairedVisit(c);
 9                 C_j.add(c_2);
10             end
11         end
12     end
13     if c.isNotAllocated then
14         i=i+1;
15         C_i.add(c);
16         if c.hasTimeDependent then
17             Visit c_2 = PairedVisit(c);
18             C_i.add(c_2);
19         end
20     end
21 end
```

Algorithm 4: Workforce Selection Module.

```
   Data: {Visits} C_i, {Workforce} K
   Result: {Workforce} ws
1 for c ∈ C_i do
2     Workforce w = bestCostForVisit(K,c,ws);
3     ws.add(w);
4 end
```

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 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

Algorithm 5: Conflict Path Detection Module.

```
   Data: {SolutionPaths} sub_sol; solutions from
         solving decomposition sub-problems
   Result: {SolutionPaths} Q; Set of conflict
           paths
 1 for {Path} s_1 ∈ sub_sol do
 2     for Path a_1 ∈ s_1 do
 3         SolutionPaths ConflictPath = null;
 4         pathConflicted=false;
 5         for s_2 ∈ sub_sol | s_2 ≠ s_1 do
 6             for Path a_2 ∈ s_2 do
 7                 if a_1.Employee = a_2.Employee
                    then
 8                     ConflictPath.add(a_2);
 9                     s_2.remove(a_2);
10                     pathConflicted = true;
11                 end
12             end
13         end
14         if pathConflicted=true then
15             ConflictPath.add(a_1);
16             s_1.remove(a_1);
17             Q.add(ConflictPath);
18         end
19     end
20 end
```

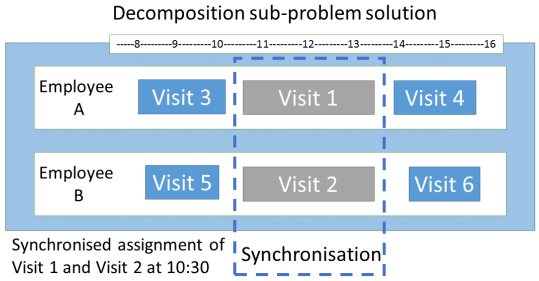
solutions and returns the set of conflicting paths Q. Basically, this process searches all sub-problem solutions and identifies all employees who are assigned to two or more paths. It then groups those conflicting paths into sub-problems to repair. This sub-problem to repair has one employee and the 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.

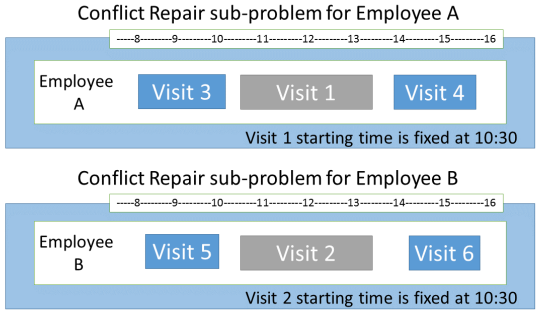
3.4 Time-dependent Activities Constraints Modification

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



(a) Decomposition sub-problem solution



(b) Conflict Repair sub-problems

Figure 1: Time dependent modification example on synchronised assignment. Sub-figure (a) shows solution from solving a decomposition sub-problem. Sub-figure (b) presents two conflict repair sub-problem solutions. Assigned time from decomposition sub-problem solution on visit with time-dependent activities (Visit 1 and Visit 2) are carried on to the later stage of process. Time window 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 cooperated in final solution or the iterative process is terminated.

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, modification 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 activities. The fixed time is applied to time window $\alpha_i = \beta_i$ where i is a time-dependent visit. 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 Figure 1(a). From the 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 $\alpha_{Visit1} = \alpha_{Visit2} = \beta_{Visit1} = \beta_{Visit2} = 10:30$. Here, we have two sub-problems to repair, presented in 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 (Castillo-Salazar et al., 2015).

4.1 WSRP Instances Set

The RDCR method was applied to the set of WSRP instances presented in (Castillo-Salazar et al., 2014; Castillo-Salazar et al., 2015). 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 (Misir et al., 2011). The Sol group are instances adapted from the Solomon dataset (Solomon, 1987). The HHC group are instances from a home

health care scenario (Rasmussen et al., 2012). Finally, the Mov group originates from instances of the vehicle routing problem with time windows (Castro-Gutierrez et al., 2011). 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 by (Castillo-Salazar et al., 2015). 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 the c (considering required skills and availability). If the number of candidate employees is less than the number of employee 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

We applied the proposed RDCR method to the 374 instances and compared the solutions obtained to the results reported for the greedy heuristic algorithm (GHI) in (Castillo-Salazar et al., 2015).

First, the related-samples Wilcoxon Signed Rank Test (Field, 2013) 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 4 showing that RDCR produced 209 better solutions 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 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.

Table 4: 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

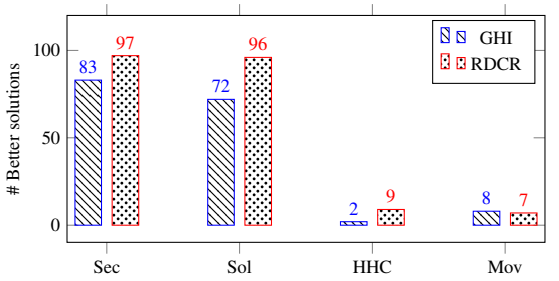


Figure 2: Number of best solutions obtained by GHI and RDCR for each dataset.

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, both 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. We can see 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.

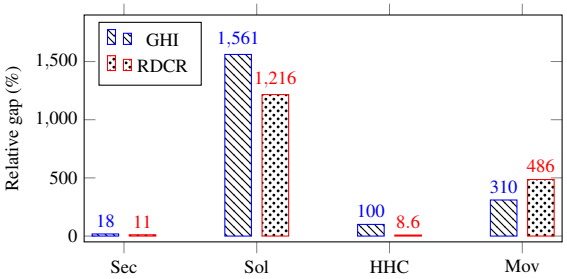


Figure 3: Average relative gap (relative to the best known solution) obtained by GHI and RDCR. The lower the bar the better, i.e. the closer to the average best known solution.

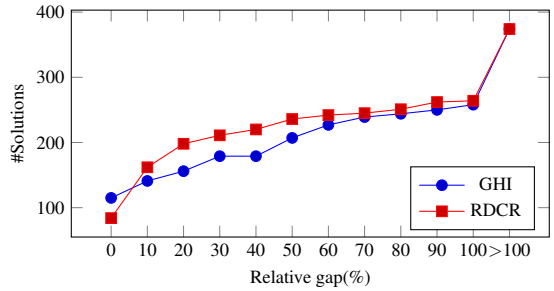


Figure 4: Cumulative distribution of GHI and RDCR solution over the relative gap.

On dataset HHC, the average relative gap of RDCR is much lower than the average gap of GHI. The results show that RDCR has 8.67% relative gap while GHI has 100.4%. 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 algorithm sorting parameters focuses 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 solver. The average relative gaps to the best known by GHI and RDCR are 315% and 36% respectively. We 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 optimally.

Figure 4 shows the cumulative distribution of RDCR and GHI solution 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. GHI provides 115 best known solutions which is better than RDCR which

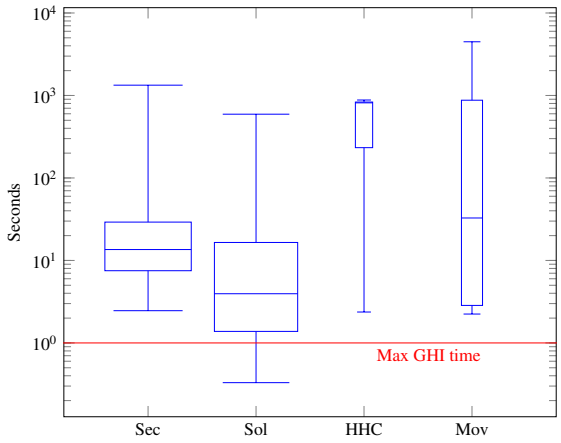


Figure 5: Box and Whisker plots showing the distribution of computational time in seconds spent by RCDR for each group of instances. The wider the box the larger the number of instances in the group. The orange straight line presents the upper bound of the computational time spent by GHI fixed to 1 second.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

provides 84 best solutions. This is 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 we set the solution acceptance rate 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 less taking less than one second on each instance. Therefore, GHI is clearly superior to RDCR in terms of computational time.

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 proposed method is called Repeated Decomposition with

Conflict Repair (RDCR). First, the method decomposes a problem into several sub-problems. Then each sub-problem is solved by an MIP solver. The solutions to the sub-problems usually include conflicting paths, i.e. two or more different sequences of visits but assigned to the same employee. These conflicting paths are tackled with a conflict repair process. Then, time-dependent activities modification is applied to tackle time-dependent activities constraints. This process maintains the layout of a solution so that a time-dependent activities constraint is valid.

The proposed RDCR approach is applied to solve four WSRP scenarios which provide a total of 374 problem instances. The experimental results showed that RDCR was 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.

Our future work is towards improving 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

We 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

Akjiratikarl, C., Yenradee, P., and Drake, P. R. (2007). PSO-based algorithm for home care worker scheduling in the UK. *Computers & Industrial Engineering*, 53(4):559–583.

Castillo-Salazar, J. A., Landa-Silva, D., and Qu, R. (2014). Workforce scheduling and routing problems: literature survey and computational study. *Annals of Operations Research*, doi:10.1007/s10479-014-1687-2.

Castillo-Salazar, J. A., Landa-Silva, D., and Qu, R. (2015). 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)*.

Castro-Gutierrez, J., Landa-Silva, D., and Moreno, P. J. (2011). Nature of real-world multi-objective vehicle routing with evolutionary algorithms. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pages 257–264.

Cordeau, J.-F., Laporte, G., Pasin, F., and Ropke, S. (2010). Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling*, 13(4):393–409.

Field, A. (2013). *Discovering Statistics Using IBM SPSS Statistics*. SAGE Publication Ltd, London, UK, 4 edition.

Laesanklang, W., Landa-Silva, D., and Castillo-Salazar, J. A. (2015). 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)*, pages 283–293.

Mankowska, D., Meisel, F., and Bierwirth, C. (2014). The home health care routing and scheduling problem with interdependent services. *Health Care Management Science*, 17(1):15–30.

Misir, M., Smet, P., Verbeeck, K., and Vanden Berghe, G. (2011). Security personnel routing and rostering: a hyper-heuristic approach. In *Proceedings of the 3rd International Conference on Applied Operational Research, ICAOR2011, Istanbul, Turkey*, pages 193–205.

Rasmussen, M. S., Justesen, T., Dohn, A., and Larsen, J. (2012). The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research*, 219(3):598–610.

Reimann, M., Doerner, K., and Hartl, R. F. (2004). D-Ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4):563 – 591.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35(2).

Xu, J. and Chiu, S. (2001). Effective heuristic procedures for a field technician scheduling problem. *Journal of Heuristics*, 7(5):495–509.