Resource constrained scheduling problems are concerned with the allocation of limited resources to tasks over time. The solution to these problems is often a sequence, resource allocation, and schedule. When human workers are incorporated as a renewable resource, the allocation is defined as the number of workers assigned to perform each task. In practice however, this solution does not adequately address how individual workers are to be assigned to tasks. This paper therefore provides mathematical models and heuristic techniques for solving this multi-period precedence constrained assignment problem. Results of a significant numerical investigation are also presented.

Keywords: Assignment problem, resource allocation, machine scheduling, meta-heuristics

1. Introduction

This paper is concerned with how individual human resources (workers) are assigned to operations (i.e. defined in the usual way) within a number of complex industrial environments and is motivated by practical relevance and the absence of existing theory. For instance, a prime example of the applicability of this problem is in a truck assembly process in Australia, where each vehicle is highly customised and produced exclusively by human labour on workstations. Typically each operation requires a minimum number of workers but additional workers may also be allocated until an upper limit is reached. For each additional worker, the processing time is reduced in a given way. Workers are also allocated to operations from specific groups (pools) of workers that service only a specific set of workstations. The minimum number of groups may be one, or as many as there are workstations. Workers may be allocated to a variety of workstations because they are multi-skilled.

This process in particular is classically described as a resource constrained permutation flowshop (RCF), which is NP-hard (see Daniels and Mazzola 1994) and whose objective is most commonly the minimisation of the makespan. From a practical viewpoint however, the allocation does not specify which specific workers should perform each operation. That is, what operations does each individual perform? To answer this question according to some measure of performance requires firstly a solution to a resource constrained scheduling problem such as the RCF. That is, as a pre-requisite, the sequence, schedule, and resource allocation for a given scheduling environment should already be determined. Hence, this research is also applicable to a variety of other machine-scheduling environments such as the jobshop, open shop, and project-scheduling problems. Many real life applications can be characterised within the framework of these problems.

Several researchers have addressed the balancing of machine workloads in flexible assembly in recent years. For example some recent papers include Sawik (1998), (Seong et al 2000), and Potts and Whitehead (2001). Their research mainly differs from this research in that the workload of fixed machines is balanced, and not the workload of individual renewable resources (workers). Our makespan is also fixed whereas theirs is not. Potts and
Whitehead in particular, simultaneously address the position (layout) of machines which is not important in our situation. Seong et al also improves efficiency and throughput in a different way by increasing and decreasing (i.e. balancing) buffer storage capacity between particular machines.

In general this problem may be classified more accurately as a form of multi-period precedence constrained non-linear assignment problem. The literature associated with related assignment problems is vast. For a review of single period problems we refer the readers to Bertsekas (1991), for precedence constrained assignment problems Sampaiboon and Yamada (1999), and for a multi-period assignment problem Cangalovic et al (1998), and Welgama et al (1999). These later two problems have not been significantly addressed to our knowledge, nor has a combination of the two been addressed before.

This problem also falls partially within the framework of a number of other classes of problems which have been found to be computationally intractable or NP-hard. These include staff rostering and labour scheduling (Alfares 2003), graph partitioning (Ahuja et al 1993), knapsack (Martello and Toth 1990), assembly line balancing problems (Ghosh S. and Gagnon R. J. 1989), and parallel machine scheduling problems (Pinedo 1995, Mokotoff 2001). All have similar aspects to that of our problem, but unfortunately none are fully able to describe and solve this problem. These classes of problem also do not provide any significant benchmarks with which to compare the results in this paper.

With particular respect to the knapsack and graph partitioning problems, we found that a variety of exact techniques such as dynamic programming, branch and bound, and graph methods have been applied. Some heuristic approaches have also been applied recently. The objective functions of these researches have all been linear, as opposed to the non-linear functions that are applicable in this research.

With particular respect to staff rostering and labour scheduling, and multi-period assignment-problems we noticed that shifts were always of fixed (equal) duration. Additional aspects such as staff preferences (for days off), and skill levels were also incorporated, which are not necessarily important in the problem in this paper. We also noted that meta-heuristics were commonly applied in the solution of these problems due to the complexity (for example Bailey et al 1997).

With particular respect to assembly line balancing problems, we found that similar types of objectives are important and similar types of heuristic approaches have also been applied (for example Chiang (1998)). The type 1 problem (i.e. minimise the number of work centres to meet the specified production requirements) differs from our problem because our equivalent work-centres are the human resources and these are fixed. The type 2 problem (i.e. allocate tasks to work-centres so that the maximum time required at any work centre is minimal) differs from our problem because it seeks to group tasks together in a work–centre, where tasks will be processed (performed) at one time in some type of sequential order. In our problem however, tasks (operation) are performed at different times across a fixed time horizon.

With respect to the parallel machine scheduling problem, each human resource in our problem may be viewed as a machine in one parallel machine work centre. Operations arrive continuously in time and must therefore be assigned to a machine for processing. The problem however, is not “dynamic”, since the operation arrival times are predetermined. However, nor is the problem “standard”, that is, where operations are perceived to be ready for processing at a given (base) starting time, and where operations are related (i.e. part of a job). It is also different because the common parallel machine scheduling objectives such as the minimisation of the makespan are not applicable (i.e. the makespan is fixed in this research).

In the next section some problem dependant properties are illustrated with respect to an example. Mathematical models are then formulated in section 3 and heuristic solution
techniques are then developed in section 4. A numerical investigation and analysis are then given in section 5. Conclusions and future research are finally discussed in section 6.

2. Problem Properties

Consider the following portion of a resource constrained flowshop solution, which is associated with a group of 10 workers servicing five workstations. In this gantt chart each operation is numbered from 1 to 15, and the total workers assigned to each operation are shown in brackets, while the completion times are shown above and below.

For this example, there are 9 stages (i.e. one for each time instance where an operation is started). For a single worker the directed graph in Figure 2 could be created from the Gantt chart in Figure 1. This graph represents all feasible assignments of the 15 operations. Operation 0 also refers to the worker not being assigned at a given stage. Note however that each node labelled zero is a different node. Alternatively, these zero operations may be removed from the graph by reassigning the arcs to and from these nodes, however this leads to a more cluttered representation (i.e. graph has 111 arcs and 16 nodes). The total number of possible (i.e. feasible) paths in the graph is 3600 and was calculated by determining the number of possible solutions to each node from the source using a breadth first search.

If we define the binary variable $X_{i,j}$ to represent whether worker $i$ is assigned to operation $j$, then the number of possible binary solutions (feasible and infeasible) is $2^{15} = 32768$, i.e. 9.1 times more per worker. Each operation $j$ can also be split into $W_j$ components, where $W_j$ is the number of workers assigned. Hence, in this graph, each node...
can be split into $W_j$ nodes with the same input and output arcs. Each additional worker may also be incorporated into this graph by an additional source node, for which the resulting graph defines all possible solutions, i.e. assignments of workers. It should be noted that a feasible solution spans the graph, that is, every node must be connected. In the original graph, input and output arcs with a total weight of $W_j$ must connect each operation node. Alternatively, each operation component node in the full expanded graph must be connected by one input and output arc. A graph generation algorithm has been devised and is explained in a later section. The full sized graph also has 68 nodes and 2007 arcs. The total number of feasible solutions for this example is as follows,

$$10 C_4^1 10 C_3^1 7 C_5^1 10 C_6^{10} 4 C_3^{10} 10 C_7^7 7 C_3^7 10 C_2^{10} 8 C_5^8 10 C_3^1 \approx 2^{71}$$

and can be determined using an iterative approach. Any problem however can have no more than $J$ stages, and this occurs when no operation overlaps another. The total number of possible assignments is given by the following expression: $\prod_{j=1}^{J} W_j$. This is an upper bound on the size of the search space which decreases as the overlapping of operations increases.

3. Mathematical Formulations

We assume that there are $i=1,...,I$ workers in the group, that are to perform $j=1,...,J$ operations on $k=1,...,K$ machines, and the time horizon of the schedule (assumed to be integer) is $t=1,...,T$. As previously mentioned, the sequence, schedule, and resource allocation for a given scheduling environment is already determined. These quantities are defined as follows and are used throughout the rest of this paper.

$E_j, C_j, P_j = C_j - E_j$: The starting, completion, and processing time respectively of operation $j$ given an assignment of $W_j$ workers.

$m_j$: The machine that operation $j$ is processed on.

We assume therefore that the input is a list of independent resources (i.e. not interested in operation precedence’s among jobs) for each group. As a result any scheduling problem concerned with the assignment of individual workers may be solved. It should be noted that each group assignment problem is solved separately. This is possible because of our assumption that workers of a particular group do not stray to other groups within a given schedule. We also assume that there are either no setup times required for workers to familiarise themselves with operations at different machines or that these set-ups are included within the processing times. Travelling times between machines are also assumed to be zero as machines assigned to each group should be close by, if not adjacent. Alternatively, travelling times should already have been incorporated into the starting solution, which is the input for our problem. Workers also perform entire operations only if not otherwise stated.

3.1. Feasibility Constraints

A feasible assignment is defined by ensuring that each operation is assigned the correct number of workers, and by ensuring that each worker is assigned to at most one
operation at any given time. That is, workers cannot be in multiple places at once. Algebraically this is represented as follows:

\[
\sum_{i \in I} X_{i,j} = W_j \quad \forall j \in J \tag{1}
\]

\[
\sum_{j \in J} X_{i,j} \leq 1 \quad \forall i \in I, t \in T \tag{2}
\]

Note that \( \Phi_t \) is the set (or list) of operations currently being processed at time \( t \) and is defined as follows: \( \Phi_t = \{ j \in J : E_j < t \leq C_j \} \). The second constraint is formulated with respect to discrete (integer) time units. That is, constraints are defined for each time point within the schedule horizon or makespan.

There are however time intervals, which include a number of discrete time points. Within these time intervals worker allocation does not change. Hence by utilising this property, an alternative and more efficient formulation is possible that reduces the number of constraints. We define the number of time intervals (stages) as \( \hat{T} \) (instead of \( T \)) with interval starting points given by \( z_i \). Constraint (2) remains the same, except for the redefinition of \( \Phi_t \) as the set of operations currently in progress during the \( t \)th interval (and not \( t \)th time), i.e. \( \Phi_t = \{ j \in J : E_j < z_i \leq C_j \} \). This leads to a reduction of \( I \cdot T - \hat{T} \) constraints. Note that this simplification also allows operations begin and end times to be non-integer values.

### 3.2. Objective Functions

A number of different measures may be used to judge the relative merits of a given assignment. It should be noted that for the project-scheduling and related problems in which there are no machines, the feasibility equations are not altered. Neither is objective function one and associated equations. Objective functions two and three are however not applicable as they are to do with movement amongst machines.

**Objective 1:** To find an assignment that balances the workload evenly amongst workers in the group. The workload for worker \( i \), and the average workload is defined as \( L_i \), and \( \bar{L} \) respectively. The objective function is therefore as follows:

\[
\text{Minimise } \sum_{i} \Delta_i \text{ where } \Delta_i = L_i - \bar{L}, \text{ and } L_i = \sum_{j} X_{i,j} P_j \quad \forall i \in I \text{ and } \bar{L} = \left( \frac{1}{I} \right) \sum_{j} P_j W_j \tag{3}
\]

The non-linearity may be removed by rewriting the unrestricted variable, \( \Delta_i \) in terms of positive variables \( \Delta^+_i \) or \( \Delta^-_i \), and by adding additional constraints as follows. The binary variable \( Y_i \) represents whether \( \Delta^+_i \) or \( \Delta^-_i \) is zero, and \( Z \) (different to \( z_i \) previously defined) is an arbitrarily large value.

\[
\text{Minimise } WLD = \sum_{i \in I} \Delta^+_i + \Delta^-_i \tag{4}
\]

\[
\Delta^+_i - \Delta^-_i = L_i - \bar{L} \quad \forall i \in I \tag{5}
\]

\[
\Delta^-_i \leq 1 - Y_i, \quad Z, \quad \Delta^+_i \leq ZY_i \quad \forall i \in I \tag{6}
\]
Complexity: The objective of balancing workload is clearly trivial if the number of workers required at each time (or interval) is equal to the number of workers in the group. That is, each worker is continuously utilised. Complexity or difficulty appears to increase however as the number of stages and level of overlapping increases.

Property: It should also be noted that minimising total workload deviation (i.e. balancing workload) is also equivalent to minimising total idle time deviation (i.e. balancing idle time). That is, \( \text{Minimise } \sum |L_i - \bar{L}| \equiv \text{Minimise } \sum |S_i - \bar{S}| \).

Proof: We note that the time spent idle by worker \( i \) is \( S_i = T - L_i \), and hence, \( \bar{S} = \frac{1}{I} \sum S_i = T - \left( \frac{1}{I} \right) \sum L_i \) by substitution. Substitution for \( S_i \) and \( \bar{S} \) into the RHS expression now completes the proof:

For this objective function, a general lower bound of zero can be defined; however, this is not particularly accurate for many situations. The workload deviation value depends on the number of workers, the operation processing times, and the number of operations and components. We propose the following equations for determining a lower bound (approximation) that is tighter than zero.

\[
\alpha_i = \sum_{j \in \Phi_i} P_j W_j, \quad \beta_i = \alpha_i / \sum_j P_j W_j, \quad \delta_i = \chi_i \beta_i, \\
\chi = \sum_{j \in \Phi_i} \left( P_j - \frac{\alpha_i}{I} W_j \right) + \left( I - \sum_{j \in \Phi_i} \frac{\alpha_i}{I} \right), \quad LB = \sum \delta_i / T. \tag{7}
\]

Note that \( \alpha_i \) is the processing required at stage \( t \), \( \beta_i \) is the percent of total processing required at stage \( t \), \( \chi_i \) is the isolated workload deviation at stage \( t \), and \( \delta_i \) is a measure of workload deviation that is proportional to the work content at the current stage. The lower bound may therefore be approximated by the average of the \( \delta_i \) values. It should be noted that the optimal solution for this example is a total workload deviation of 5.6 and was determined by the CPLEX optimiser. The lower bound approximation is 3.93 as shown by the following calculations.

Table 1. Lower bound (approximation) calculations for the example

<table>
<thead>
<tr>
<th>Step</th>
<th>Workers</th>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>( \alpha/I )</th>
<th>( \chi )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>0.07</td>
<td>4</td>
<td>48</td>
<td>3.29</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>0.16</td>
<td>9.6</td>
<td>54</td>
<td>8.88</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>0.23</td>
<td>13.4</td>
<td>43.2</td>
<td>9.91</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>0.06</td>
<td>3.6</td>
<td>50.4</td>
<td>3.11</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>0.14</td>
<td>8.1</td>
<td>16.8</td>
<td>2.33</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>0.21</td>
<td>12.5</td>
<td>21</td>
<td>4.49</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>0.03</td>
<td>1.5</td>
<td>21</td>
<td>0.54</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>0.08</td>
<td>4.8</td>
<td>32</td>
<td>2.63</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>0.02</td>
<td>0.9</td>
<td>12.6</td>
<td>0.19</td>
<td></td>
</tr>
</tbody>
</table>

**Objective 2:** To minimise the total movement of workers so that travelling distances or travelling times are (implicitly or explicitly) minimised. Firstly it should be noted that the
number of times worker $i$ moves during the schedule is a value between zero and $\sum_j X_{i,j}$. If we want to explicitly minimise movement in terms of total distance travelled, then we need to define the distance between each machine, $\text{dist}_{k,k'}$ where $k,k' \in K$. Note that if $k = k'$ then $\text{dist}_{k,k} = 0$. We also need to define the position of a worker at each time. This is accomplished by defining the variable $\text{pos}_{i,t} \in (0,J)$ as the position of worker $i$ at time $t$. It is calculated by the following equation, $\text{pos}_{i,t} = \sum_{j \in E_{t}} m_j X_{i,j}$, where $m_j$ is the machine required by operation $j$. The objective function can therefore be written as follows:

$$\text{Minimise} \sum_i \sum_{t \in p} \text{dist}_{\text{pos}_{i,t_{-}}, \text{pos}_{i,t+1}}$$

(8)

Note that the initial position of resources is assumed to be of no consequence, i.e. not included in (8). However this additional aspect is easily incorporated if necessary, and may be of particular interest when a new production schedule is tagged onto the end of the previous one. If $\text{dist}_{k,k'} = 1 \, \forall k,k': k \neq k'$, then the problem is also equivalent to minimising total movement. An alternative approach might be to define a binary variable to represent whether worker $i$ moves at the end of time interval $t$. To determine $z_{i,t}$ we notice that movement occurs at the end of time $t$ if $\text{pos}_{i,t} \neq \text{pos}_{i,t+1}$, i.e. there exists $j, j': X_{i,j} = 1, X_{i,j'} = 1, m_j \neq m_{j'}, E_{j'} \geq C_{j'}$. No linear constraint however can be defined for this condition, however, the model could still be formulated in the following way:

$$\text{Minimise} \sum_i \left( \sum_{t} \min |\text{pos}_{i,t} - \text{pos}_{i,t+1}|, 1 \right) \equiv \sum_i \left( \sum_{t} \min \gamma_{i,t}^{+} - \gamma_{i,t}^{-}, 1 \right)$$

(9)

$$\gamma_{i,t}^{+} - \gamma_{i,t}^{-} = \text{pos}_{i,t} - \text{pos}_{i,t+1} \quad \forall i,t$$

(10)

$$\gamma_{i,t}^{+} \leq 1 - Y_i \times 1000 \quad , \quad \gamma_{i,t}^{-} \leq Y_i \times 1000 \quad \forall i,t$$

(11)

The lower bound can again be assumed as zero for this objective. A more accurate approximation however may be useful as it was for the first objective.

For this objective we note the following properties. Minimising total distance travelled is equivalent to minimising total travelling time if travelling time is linearly proportional to the distance. Minimising distance travelled does not necessarily ensure that movement is minimised. The proof of this property is associated with the fact that a large number of small movements may minimise distance but will increase the number of movements above that of another assignment which has a small number of large movements (i.e. distances). Note also that movement is automatically implied when any distance is travelled; however movement does not specifically define the distance travelled.

For the example, the best solution (under this objective) was found to be 36, with movement per worker 6, 3, 2, 3, 4, 3, 3, 2, 5, and 4 respectively. The workload deviation is much higher however with a value of 52. The assignment problem with this second objective function may be alternatively solved (exactly) as a type of network flow problem (degree constrained minimum spanning tree) using the precedence graph explained in the introduction.

The graph may be constructed by Algorithm 1 (see Appendix) which assumes that each operation is already split into its $W_j$ component parts. Hence $J$ is not the number of
operations, but the number of components. This also means that each component is viewed as an operation with only one component. The first set of arcs required connects the workers to the components. The second set of arcs is the precedence arcs between operation components. As an alternative, another algorithm may be used if the operations have not been previously split. However this is much more difficult to implement efficiently. We suggest that all operations are split and the data reformatted beforehand using Algorithm 2 (see Appendix).

This reformattting is also required by the heuristic approaches to be later discussed. The number of nodes is the number of workers plus the number of components. The number of arcs however is proportional to the particular precedence constraints of a problem. However, an upper bound on the number of arcs may be determined by the expression $IJ + J(J-1)$ which occurs when each component occurs serially. The lower bound is $IJ$ which occurs when each component occurs at the same time. This results in a standard bi-partite matching problem. This assignment problem may also be viewed as a type of K-TSP (Ahuja et al 1993). That is, K salesman, all starting from the same starting node, must visit all other nodes in the graph. The only difference is that some nodes are only reachable if the salesman has visited certain other nodes. This is because of the precedence constraints.

**Objective 3:** To minimise the number of machines a worker is assigned to, to retain a high level of specialisation. We define the number of operations performed by worker $i$ on machine $k$ as $\theta_{i,k}$. It should be noted that for any given worker $i$, $\sum_k \theta_{i,k} = \sum_j X_{i,j}$. The number of machines that worker $i$ is assigned to is therefore, $\sum_{k: \theta_{i,k} \geq 1} 1 \equiv \sum_k \min 1, \theta_{i,k}$ and the objective function is therefore as follows:

$$\text{Minimise } \sum_i \left( \sum_k \min 1, \theta_{i,k} \right) \quad \text{where } \theta_{i,k} = \sum_{\forall j: m_{j,k}} X_{i,j} \quad \forall i \in I, k \in K$$

This is probably the least important objective proposed. However this objective may be important for situations where the number of machines looked after by a group of workers is high and spaced a fair distance from one another. Therefore there appears to be some similarity or equivalence between this objective function and that of minimising movement or distance travelled. When dealing with manufacturing industries (particularly the truck assembly process mentioned) the number of machines per group is generally small. However there may be environments where only one group exists and this objective function is aimed at this occurrence.

### 3.3. A Variation of the Original Problem

We remove the assumption that workers perform entire operations. Therefore, a worker may perform part of an operation before moving to another machine to perform a different operation. We change the main binary variable to represent whether worker $i$ is assigned to operation $j$ during time interval $t$ instead (i.e. $X_{i,j,t}$).

#### 3.3.1. Feasibility Constraints

The feasibility constraints become the following:

$$\sum_i X_{i,j,t} = W_j \quad \forall j, t : E_j < t \leq C_j$$

(13)
\[
\sum_{i,j,t} X_{i,j,t} \leq 1 \quad \forall i \in I, t \in T \\
X_{i,j,t} = 0 \quad \forall i, j, t : t \leq E_j \mid t > C_j
\]

(14)  

(15)

### 3.3.2. Objective Functions

**Objective 1:** The only change occurs in the workload constraint:

\[
L_i = \sum_t \left( \sum_{j \in \Phi_i} X_{i,j,t} \right) \quad \forall i \in I
\]

(16)

Alternatively, the following model can be used where \( X_{i,t} \) is defined instead as a binary variable that represents whether worker \( i \) is assigned to time interval \( t \). Note that the duration of the time interval is \( DUR_i \), and the number of workers required at time interval \( t \) (i.e. over all operations in progress during the time interval) is \( W_t \). For this objective function it is not important to know which operation a worker is assigned to, because each worker assigned during the interval has the same workload.

\[
\text{Minimise} \sum_t |L_t - \bar{E}| \quad \text{where } L_t = \sum_i X_{i,t} DUR_i \quad \forall i
\]

(17)

s.t. \[
\sum_i X_{i,t} = W_t \quad \forall t
\]

(18)

\[
X_{i,t} = 0 \text{ or } 1 \quad \forall i, t
\]

The number of time intervals, the duration of each, and the number of workers required must however be determined using an iterative algorithm. At each step of the algorithm a new time point is determined as follows: \( t' = \min_{j : E_j > C_j} E_j \mid C_j \). The duration of the interval is therefore \( t' - t \) and the number of workers required is \( \sum_{j : E_j > C_j} W_j \). Before the next step, the new time point is set as \( t' \) and the process is repeated until \( t = T \). For the main example, there are 14 time intervals, which start at the following times, 0, 10, 17, 25, 33, 42, 45, 51, 54, 55, 66, 71, 75, 79, and whose duration is the following, 10, 7, 8, 8, 9, 3, 6, 3, 1, 11, 5, 4, 4, 3. The number of workers required in each time interval is also as follows: 4, 8, 5, 10, 9, 3, 10, 6, 3, 10, 6, 7, 5, 3.

**Objective 2:** Modify the equation for the position of a worker to the following:

\[
pos_{i,t} = \sum_{j \in \Phi_i} m_j X_{i,j,t}
\]

(19)

**Objective 3:** The number of operations performed by worker \( i \) on machine \( k \) is modified as follows:

\[
\theta_{i,k} = \sum_t \sum_{j \in \Phi_i, m_j = k} X_{i,j,t} \quad \forall i \in I, k \in K
\]

(20)
4. Heuristic Solution Approaches

To solve practical sized problems an analytical approach may not be feasible due to the complexity of the problem. For instance, the solution of the main example was attempted by CPLEX and computation was in excess of 2 hours even for such a simple problem. The development of heuristics is therefore necessary. Firstly however we propose adequate solution generation techniques for any heuristic approach that may be developed.

4.1 Constructive Algorithms for Solution Generation

Algorithm 3 (see Appendix) can be used to obtain a good starting solution for the objective of balancing workload. The logic behind this constructive algorithm is that at each time step the operation component with the largest processing requirements is assigned to the free worker with the lowest workload assigned so far. The solution is built from time zero or alternatively in a backwards manner from the schedule makespan. In this algorithm the set of available workers at time \( t \) is the ordered set \( \Omega \), and the set of schedulable operations at time \( t \) is the ordered set \( \Psi \). These ordered sets could be implemented as priority queues. In particular, an efficient data structure for priority queues is a heap. A heap is a complete binary tree, in which the priority value of each node is greater than or equal to the priority value of its children. The insertion or removal of elements is \( O(N \log N) \) and has the effect of sorting.

This algorithm however produces only one solution of good quality. It should also be noted that at some time steps it might also be better to assign some operations to workers that do not have the smallest workload thus far. Hence to obtain more than one solution from this algorithm, the order in which workers are selected is changed randomly (i.e. say 10% of time) in the following way:

\[
\text{if } \text{rnd} \ 0.1 \leq 0.1 \ \text{then insert pair } \left(i, \text{rnd} \left( \min_{u \in \Omega} L_u, \max_{u \in \Omega} L_u \right) \right) \ \text{else insert } i, L_i
\]

Alternatively, if a solution is to be rebuilt from a given time \( t \) within the schedule horizon, then the only change required is in the initialisations. The required modifications are as follows, where \( F_i \) is the free time of worker \( i \).

\[
L_i = \sum_{j \in E_i} P_j X_{i,j} \quad \forall i
\]

\[
F_i = C_j \quad \forall i \ \exists j \in J : E_j < t \leq C_j, \text{otherwise } F_i = t
\]

If we can assign a worker at every integer time instant (i.e. the problem variation), then the absolute lower bound of zero may be reachable. Our solution generation algorithm can be used to obtain a solution with only the following modifications in the initialisation.

\[
F_i = t \quad \forall i \ \forall j, L_i = \sum_{j \in J} X_{i,j} \quad \forall j \in J : E_j \leq t < C_j \quad (23)
\]

For the following heuristic approaches, the number of operations is set as the number of components. Hence the decision variable can be changed from the binary variable \( X_{i,j} \) to \( X_j \in 1, I \). The constructive solution generation algorithm (CSGA) is also changed minutely to accommodate this difference. Constructive Algorithm 4 (see Appendix) may also be used to construct a good solution for the second objective of minimising total movement and also with small modification to the objective of minimising total distance travelled. The
logic behind this constructive algorithm is that at each step, movement is minimised by keeping workers at their current locations if possible. If not possible then the worker is moved to another location and is assigned to an operation component randomly. This strategy is performed at each step iteratively until all operation components have been assigned workers.

4.2. Simulated Annealing Approach

The local search operators described in this section are utilised within the standard SA control structure which contains an outer while loop for the number of temperature steps, and an inner while loop for the number of iterations at each temperature step. The local search operators in particular are applied within the inner loop to create a candidate solution. The candidate solution is evaluated, and the usual SA acceptance criteria are applied (i.e. a solution is accepted if strictly better or according to a small probability that is proportional to the current temperature). The new solution is compared to the best so far (stored if better) and the iteration counter is incremented. At the end of each temperature step the number of steps is incremented and a new temperature value is obtained by multiplying the previous value by the temperature reduction parameter.

The CSGA operators are used as the basis for our first simulated annealing application (SA1). Because CSGA always returns a feasible solution, the objective function can be computed with less computational effort. The operator is used to firstly define a good starting solution. This solution is built from scratch (i.e. time 0) and with no randomness incorporated. The operator is then used to create a new candidate solution by rebuilding the current solution from a randomly chosen time within the schedule horizon. The algorithm is also the random version as opposed to the deterministic version used to create the initial starting solution.

Our second SA approach (SA2) is essentially a hybrid of SA and TS. The CSGA operator is again applied, but for each instant of time that an operation can be begun (i.e. instead of just once at a randomly selected time). Our neighbourhood is therefore the number of these time points. A tabu list is not used because we do not directly change the solution deterministically. It should be noted that the value of time that will give the greatest benefit when the solution is rebuilt by the CSGA operator (and for the workload deviation objective) is the largest $\delta_i$ values (from the lower bound approximation section). This makes sense if one considers the example. The second, third and sixth time periods are most important in terms of total workload.

An exchange operator is used as the basis for our third simulated annealing algorithm (SA3). That is two components, $j$ and $j'$ are chosen and the workers assigned to each are exchanged, i.e. $X_j \leftrightarrow X_{j'}$. Two possible redundancies that are to be avoided occur when $j = j'$ or when $X_j = X_{j'}$. A third possible redundancy may be also avoided and occurs when two workers are swapped between two components of the same operation or between two components with the same starting and processing time. Infeasibilities may occur in the solution when the exchange operator is applied. The infeasibilities are caused in particular by the violation of constraint (2). These infeasibilities however are not removed, but rather penalised in the objective function as follows, where the penalty value lambda is arbitrarily set as 1000.

$$\text{infeasibilities} = \sum_{i} \sum_{j} \sum_{\mathcal{E}:X_j = i} 1 \quad \text{and penalty} = \lambda \times \max_0, \text{infeasibilities} - 1 \quad (24)$$

4.3. Tabu Search Approach

The neighbourhood exchange operator used in the SA approach is also used as the basis for two tabu search approaches (TS1) and (TS2). The full neighbourhood for this
operator however is rather large, i.e. $J - 1 \approx O(J^2)$, but is nonetheless used in (TS1). An alternative is to choose $j$ randomly and then compare all possible $j'$, i.e. $X_j \leftrightarrow X_j' \forall j' \neq j, X_j \neq X_j'$. The neighbourhood is therefore of size $O(J)$ and this approach is taken in (TS2). Pairs of operations (used in an exchange) are inserted into a tabu list. As with the SA approaches, we have used a standard TS control structure. This contains an outer while loop for the number of iterations. Within this loop the (chosen) neighbourhood of the current solution is fully investigated. That is, each solution is evaluated, compared and stored if it is the best so far and not tabu. At the end of the iteration, the best move is finally made, the exchange is added to the tabu list, and the iteration counter is then incremented.

4.4. Evolutionary Algorithm Approach

An evolutionary approach was found to be unsuitable for this problem because the crossover operators that were investigated were found to be inappropriate. The reason for this is that the crossover operators are generally not sufficient for local search, particularly in a problem that is so heavily constrained. Since the starting population is of such high quality, an efficient local search routine is only really required. We found no benefit from any crossover operator applied. All resulting child solutions were inferior. However, crossover operators that were investigated for this problem included the following:

CX1: Assign attribute to child that is common in each parent, i.e. $X_j^{child} = i$ iff $X_j^{mother} = X_j^{father} = i$. Assign all other $X_j$ constructively.

CX2: Assign attribute to child that is common in each parent, i.e. $X_j^{child} = i$ iff $X_j^{mother} = X_j^{father} = i$. Assign all other $X_j$ randomly from either parent, i.e. $X_j^{child} = X_j^{mother}$ if $rnd \ 0,1 < 0.5$, otherwise $X_j^{child} = X_j^{father}$.

CX3: Standard one point crossover with the binary solution representation. Another alternative is to exchange (swap) entire portions of the solution, for example: $X_{i,j}^{child} = X_{i,j}^{mother}$ \forall \ j \neq i', X_{i,j}^{child} = X_{j}^{father} \forall \ j, i = i'$.

The problem with these crossover operators in particular is that an operation may be feasibly assigned to a worker at one time instant, however this assignment will overlap another assignment (already fixed) at a different time instant. In this scenario, a different worker would need to be assigned to the operation, but in some cases this is not sufficient, as there are no other workers to assign.

5. Numerical Investigation

The test problems for our numerical investigation are explained below, and consist of randomly generated problems and real instances from industry. The heuristics were applied to each test problem for the objectives of balancing workload, minimising movement and minimising distance travelled. A multi-objective function of balancing workload and minimising distance travelled was also used. This objective function is just the sum of the two with equal weights.
5.1. Test Problems

**Real:** We investigated a truck assembly problem with 59 stations and 93 workers. The workers are split amongst 5 groups as follows, 12, 25, 22, 17, 17 and the stations are also assigned to the same groups in the following numbers, 3, 13, 17, 10, 16. We investigate a RCF solution consisting of 15 jobs or 885 operations. Hence we are solving 5 independent problems as follows: (12,171,3,87) (25,569,13,62.63) (22,538,17,65.25) (17,412,10,77.53) (17,391,16, 64.91), where each problem is classified by (I/J/K/T). Note that the average workload (in hrs) for each problem is as follows: 41.17, 26.82, 36.64, 31.00, 37.24.

**Randomly generated problems:** We generated test problems consisting of 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, and 200 components. For each of these, 10 instances were generated (i.e. a total of 110 test problems) by randomly creating component start and end times using the following: \( E_j = \text{rnd} \ 0,40 \), \( C_j = E_j + \text{rnd} \ 0,5 \) and such that \( C_j - E_j > 0 \). A single worker performed each component, so that the total number of workers I is defined by the maximum number of components performed at any time during the time interval.

5.2. Parameters

After an initial experimentation phase in which different parameters were trialled, we observed that the following parameters gave the best performance in terms of convergence and solution quality. These parameters also gave the best comparison of performance among the different heuristics.

SA1: starting temperature 1, temperature steps 100, iterations 200, tfactor 0.9
SA2: starting temperature 1, temperature steps 100, iterations 100, tfactor 0.9
SA3: starting temperature 1, temperature steps 100, iterations 1000, tfactor 0.9
TS1: iterations 20, tabu list 20
TS2: iterations =100 000/J, tabu list=20

In particular, SA1 required more iterations (at each temperature step) than SA2 because more applications of the CSGA operator were made at each iteration of SA2 (i.e. CSGA operator applied for every value of time). SA3 however required more iterations than both SA1 and SA2 because changes made by the exchange operator could be performed with less computational effort, and had less immediate impact on the solution. For the selection of the other SA parameters (i.e. starting temperature, temperature steps, and temperature reduction factor), a “greedy” approach was essentially taken. The approach is “greedy” because (with these parameters) the SA algorithm is directed to make moves (more often) that allow strictly better solution to be reached, as opposed to making moves that do not improve upon the current solution. The parameters however are such that, a reasonable number of uphill moves are still made in the early stages of the algorithm. Alternatively a higher temperature and/or smaller reduction factor was investigated as this allowed more moves to be accepted, thereby improving the chance of escaping locally optimal solutions. However this drastically increased computation time and there was no guarantee that the solution would be better given the additional effort.

The TS1 iteration limit was chosen as 20 because very little improvement was observed after this limit had been reached, and significant improvements were still occurring beforehand. The number of iterations for the TS2 heuristic however was chosen so that the number of function evaluations was the same as the SA3 algorithm. We investigated the effect of changing the tabu list size, but we did not observe any significant difference in the results, therefore we chose a value of 20 arbitrarily for the results in the remainder of this paper.
5.3. Results

All results below were generated on a Pentium 3 850 MHz computer. The relative error (RE) values are determined with respect to the lower bound approximation. TS1 was not used for the truck assembly problems because the computational times were too prohibitive. A tighter lower bound approximation has not been devised for the objective of minimising total movement and hence no relative error values are displayed in Table 4, 5 and 6. The SA1 and SA2 approaches are also not displayed after Table 3 as the CSGA operator was found to be very ineffective on these problems.

<table>
<thead>
<tr>
<th>#</th>
<th>LB</th>
<th>Starting Solution</th>
<th>SA1 RE</th>
<th>SA2 RE</th>
<th>SA3 RE</th>
<th>TS2 RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.63</td>
<td>2.86</td>
<td>3.55</td>
<td>0.72</td>
<td>1.85</td>
<td>0.15</td>
</tr>
<tr>
<td>2</td>
<td>0.06</td>
<td>4.92</td>
<td>2.93</td>
<td>2.93</td>
<td>50.13</td>
<td>2.20</td>
</tr>
<tr>
<td>3</td>
<td>0.06</td>
<td>14.03</td>
<td>228.26</td>
<td>9.49</td>
<td>1.50</td>
<td>154.07</td>
</tr>
<tr>
<td>4</td>
<td>0.06</td>
<td>1.72</td>
<td>27.68</td>
<td>0.94</td>
<td>1.43</td>
<td>14.72</td>
</tr>
<tr>
<td>5</td>
<td>0.05</td>
<td>5.95</td>
<td>121.47</td>
<td>3.31</td>
<td>1.20</td>
<td>67.15</td>
</tr>
<tr>
<td>Averages</td>
<td></td>
<td></td>
<td>93.18</td>
<td>1.56</td>
<td>57.243</td>
<td>44.56</td>
</tr>
</tbody>
</table>

Table 3: Workload balancing results for randomly generated problems

<table>
<thead>
<tr>
<th>Test Problem</th>
<th>Starting Solution</th>
<th>SA1 AVG (min)</th>
<th>SA1 CPU</th>
<th>SA2 AVG (min)</th>
<th>SA2 CPU</th>
<th>SA3 AVG (min)</th>
<th>SA3 CPU</th>
<th>TS1 AVG (min)</th>
<th>TS1 CPU</th>
<th>TS2 AVG (min)</th>
<th>TS2 CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>5.503</td>
<td>0.02</td>
<td>0.64</td>
<td>0.07</td>
<td>0.61</td>
<td>0.06</td>
<td>0.78</td>
<td>0.00</td>
<td>1.22</td>
<td>0.20</td>
<td>0.78</td>
</tr>
<tr>
<td>11-20</td>
<td>11.6</td>
<td>0.07</td>
<td>0.86</td>
<td>0.44</td>
<td>0.26</td>
<td>0.19</td>
<td>0.05</td>
<td>0.10</td>
<td>0.81</td>
<td>0.22</td>
<td>0.09</td>
</tr>
<tr>
<td>21-30</td>
<td>19.19</td>
<td>0.14</td>
<td>3.44</td>
<td>1.24</td>
<td>2.04</td>
<td>0.39</td>
<td>-0.08</td>
<td>0.06</td>
<td>1.07</td>
<td>0.37</td>
<td>0.20</td>
</tr>
<tr>
<td>31-40</td>
<td>22.05</td>
<td>0.21</td>
<td>5.21</td>
<td>2.21</td>
<td>3.23</td>
<td>0.59</td>
<td>0.05</td>
<td>0.18</td>
<td>0.39</td>
<td>0.55</td>
<td>0.46</td>
</tr>
<tr>
<td>41-50</td>
<td>29.27</td>
<td>0.32</td>
<td>10.14</td>
<td>4.17</td>
<td>7.82</td>
<td>0.94</td>
<td>-0.14</td>
<td>0.45</td>
<td>0.74</td>
<td>0.86</td>
<td>0.77</td>
</tr>
<tr>
<td>51-60</td>
<td>28.15</td>
<td>0.39</td>
<td>11.19</td>
<td>5.60</td>
<td>7.75</td>
<td>1.20</td>
<td>0.04</td>
<td>0.80</td>
<td>0.64</td>
<td>1.09</td>
<td>0.86</td>
</tr>
<tr>
<td>61-70</td>
<td>29.11</td>
<td>0.50</td>
<td>12.74</td>
<td>7.51</td>
<td>9.23</td>
<td>1.61</td>
<td>0.23</td>
<td>1.52</td>
<td>0.67</td>
<td>1.46</td>
<td>1.16</td>
</tr>
<tr>
<td>71-80</td>
<td>40.14</td>
<td>0.61</td>
<td>15.74</td>
<td>10.71</td>
<td>11.56</td>
<td>2.04</td>
<td>-0.09</td>
<td>2.46</td>
<td>0.51</td>
<td>1.85</td>
<td>1.15</td>
</tr>
<tr>
<td>81-90</td>
<td>44.91</td>
<td>0.77</td>
<td>21.12</td>
<td>15.02</td>
<td>14.95</td>
<td>2.70</td>
<td>0.08</td>
<td>4.16</td>
<td>0.31</td>
<td>2.443</td>
<td>1.247</td>
</tr>
<tr>
<td>91-100</td>
<td>45.79</td>
<td>0.87</td>
<td>21.84</td>
<td>18.30</td>
<td>16.85</td>
<td>3.12</td>
<td>0.26</td>
<td>12.14</td>
<td>0.19</td>
<td>2.30</td>
<td>2.19</td>
</tr>
<tr>
<td>101-110</td>
<td>50.32</td>
<td>2.58</td>
<td>36.90</td>
<td>34.00</td>
<td>30.15</td>
<td>10.48</td>
<td>0.15</td>
<td>79.75</td>
<td>1.20</td>
<td>9.82</td>
<td>2.87</td>
</tr>
<tr>
<td>Average</td>
<td>29.64</td>
<td>12.71</td>
<td>9.49</td>
<td>0.12</td>
<td>0.70</td>
<td>1.06</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From table 2, it is clear that SA3 is clearly superior on the industry problems with an average RE of 1.45. However the TS2 approach is nearly as good, for example, it gives better solutions on two of the five problems. TS2 however is not as consistent overall, for example on problem 3 and 5 the results are quite different to SA3. SA1 and SA3 gave good results on problem 1 but failed to obtain good solutions for the other problems. SA1 and SA3 computation times were also far too high.

From table 3, similar conclusions can be made however the gap between SA3 and TS2 is smaller. On these randomly generated problems, SA3 was also able to improve upon the
lower bound approximation on 50% of the problems, as were the other approaches on a smaller scale. TS1 was also superior to TS2 and took less computation time on the majority of the problems even though the neighbourhood used was larger. However on the larger problems (81-110), the reverse occurred, i.e. computation times were significantly greater. SA1 and SA2 were also more effective on the randomly generated problems than on the industry problems.

Table 4. Minimising total movement of workers results

<table>
<thead>
<tr>
<th>Group</th>
<th>Starting Solution</th>
<th>SA3</th>
<th>TS2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TM RE CPU (min)</td>
<td>TM RE CPU (min)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>101 80 -0.21 6.95</td>
<td>88 -0.128 6.26</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>797 696 -0.126 81.6 704 -0.117 75.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>662 501 -0.24 75.03 533 -0.195 69.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>569 473 -0.169 38.2 484 -0.15 34.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>437 319 -0.27 45.51 343 -0.215 41.65</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For the alternative objective of minimising total movement, SA3 is on average 20% better than the starting solution, while TS2 is 16% better. This improvement over the initial starting solution is much less than for the workload balancing scenario where all of the heuristics improved upon the starting solution by at least 40% and routinely by 70-100%.

Table 5. Minimising total distance travelled results

<table>
<thead>
<tr>
<th>Group</th>
<th>Starting Solution</th>
<th>SA3</th>
<th>TS2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TD RE WLD CPU (min)</td>
<td>TD RE WLD CPU (min)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1222 96.6 -0.92 151.557 6.98 193.2 -0.842 133.917 6.35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4205.3 889.5 -0.79 342.729 81.816 1659.5 -0.61 335.249 75.71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3761.7 347.2 -0.91 294.07 74.51 785.8 -0.79 237.104 69.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2462.3 482.9 -0.8 200.21 38.31 649.5 -0.74 188.695 34.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2206.6 90.9 -0.96 232.85 45.5 211 -0.9 156.991 41.5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For the minimisation of distance travelled, SA3 and TS2 respectively, improved considerably upon the starting solution by 87.6% and 77.6% on average. Accompanying workload deviation values are also given in table 5. These values show that minimising the distance travelled unbalances the workload considerably. As a consequence we investigated a multi-objective function that minimises distance travelled and workload deviation (note equal weights in objective function). We looked at this combination only, because we perceived that it was more likely to occur and be of interest to industry. We also wanted to further investigate how close the respective components were in comparison to the results of the single objective problem. From the results in table 6, we can see that better results were obtained for the distance travelled objective than in table 5. Hence this means that if we wish to minimise total distance travelled then we should use the multi-objective function that also takes into account balancing workload. Note also that for this multi-objective function, SA3 and TS2 respectively, improved considerably upon the starting solution by 81% and 73% on average.
Table 6. Results of truck assembly problem for multi-objective function of minimising distance travelled and balancing workload.

<table>
<thead>
<tr>
<th>Group</th>
<th>Start Soln</th>
<th>SA3 Final</th>
<th>RE</th>
<th>TD</th>
<th>WLD</th>
<th>TS2 Final</th>
<th>RE</th>
<th>TD</th>
<th>WLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1373.56</td>
<td>282.813</td>
<td>-0.79</td>
<td>131.256</td>
<td>288.85</td>
<td>193.2</td>
<td>95.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4458.03</td>
<td>1214.92</td>
<td>-0.73</td>
<td>872.191</td>
<td>342.729</td>
<td>2008.19</td>
<td>-0.55</td>
<td>1688</td>
<td>320.187</td>
</tr>
<tr>
<td>3</td>
<td>4055.78</td>
<td>460.853</td>
<td>-0.89</td>
<td>166.78</td>
<td>294.074</td>
<td>966.992</td>
<td>-0.76</td>
<td>769.99</td>
<td>196.99</td>
</tr>
<tr>
<td>4</td>
<td>2662.52</td>
<td>667.932</td>
<td>-0.75</td>
<td>467.717</td>
<td>200.215</td>
<td>794.619</td>
<td>-0.7</td>
<td>608.9</td>
<td>160.719</td>
</tr>
<tr>
<td>5</td>
<td>2439.45</td>
<td>259.228</td>
<td>-0.894</td>
<td>26.377</td>
<td>232.851</td>
<td>359.648</td>
<td>-0.85</td>
<td>232.6</td>
<td>127.048</td>
</tr>
</tbody>
</table>

6. Conclusion

This paper addressed a multi period precedence constrained assignment problem that was found in practice. We firstly reviewed the literature for this problem and found many classes of similar problems. However as far we can tell this problem situation has not been significantly addressed before. A number of different objective functions were then proposed and investigated. The reason for this is that different industries have differing requirements. Alternatively, they may have a combination of measures. Mathematical models for this problem were also presented for completeness, and as alternative avenues for analytical solution techniques. However because of the complexity of the problem, heuristic solution techniques were required. Some problem dependant properties were also investigated as were representational issues such as directed graph representations, construction, and usage.

A number of heuristics were proposed for this problem and evaluated on a number of real and randomly generated problems and for several alternative objective functions. From the numerical analysis the best heuristic regardlessly was found to be simulated annealing with an exchange operator. The exchange operator was found to be an efficient local search tool not only for simulated annealing but also for tabu search. The tabu search approaches were also found to give good solutions, but in comparison with SA, they were found to be generally more computationally expensive, and did not always reach as good solutions. However on some occasions one of the two proposed tabu search approaches outperformed SA. Evolutionary strategies for this problem were also investigated but no suitable crossover algorithms were found that could maintain or repair solution feasibility. An EA approach may in fact be suitable, but further research is required here.

Two efficient constructive solution generation algorithms (CSGA) were also proposed and used to determine starting solutions of high quality. They were also used as the basis for a heuristic and also performed adequately on small problems. They however did not have the ability to reach as good solutions on larger problems. The lower bound approximation (for the first objective of minimising total workload deviation) was also found to be reasonably accurate. In some circumstances however, the best solution obtained was fractionally better, or below the lower bound approximation. This occurred in the majority of cases for the randomly generated problems and seldom on the industry problems. We conclude that the approximation is more accurate in situations where overlapping is more frequent as occurs in industry. In particular this occurred 56 times out of 115 for the best heuristic but with only an average RE of 0.3 better. Hence the lower bound approximation may still be used but the value should be scaled by a reduction factor, particularly for problems where overlapping does not occur so much.

The objective of minimising total movement was found to be much harder for the heuristics to solve than the balancing workload problem, or for minimising distance travelled as shown in the tables of results. The reason for this is that very little change occurs in the movement objective function when an exchange is made. For the other objective functions, an exchange has a large effect on the objective function value, and hence it is easier to see which
moves are more beneficial. For objectives other than balancing workload it is not known how close our solutions are to the optimal solution and this is because a tighter lower bound approximation was not developed. From a practical point of view however, this research should allow industry to assign their workers (resources) in an equitable manner throughout their production environments, with reasonably small computational effort. Future research may consider the determination of more efficient lower bounds for the different objective functions proposed and the investigation of the variant problem. How problem complexity or difficulty increases with respect to input data is also a source of further research which was outside the scope of this paper.

7. References


Bertsekas D. (1991), Linear network optimisation: algorithms and codes. MIT Press, USA.


8. Appendix

Algorithm 1: Graph construction
Input: $I, J, E, C, m$
Begin
Graph $G = V, A$ \; $|V| = I + J$
Add arcs of weight 0 between $v_i, v_{i+j}$ \; $\forall i \in I, j \in J$
Add arcs between $v_{i+j}, v_{i+j'}$ \; $\forall j, j' \in J, j \neq j', C_j \leq E_j$ with weight 0 if $m_j = m_{j'}$
End

Algorithm 2: Data formatting
Input: $I, J, E, C, W, m$
Begin
$\lambda = 0$
for $j = 1, \ldots, J, k = 1, \ldots, W_j$ do $\lambda = \lambda + 1$; \; $E'_j = E_j$; \; $C'_j = C_j$; \; $m'_j = m_j$; \; $W'_j = 1$
Redefine the following: $J = \lambda$; \; $E = E'$; \; $C = C'$; \; $m = m'$; \; $W = W'$
End

Algorithm 3: CSGA(WLD)
Initialisation: $t = 0$; \; $L_i = F_i = 0$ \; $\forall i$; \; $\Psi = \Omega = \emptyset$
while ($t < T$) begin
$\Psi = \{ j, P_j \mid \forall j \in J : E_j = t \}$; \; // Note that $u, P_u < v, P_v$ iff $P_u > P_v$
$\Omega = \{ i, L_i \mid \forall i \in I : F_i \leq t \}$; \; // Note that $u, L_u < v, L_v$ iff $L_u < L_v$
while ($|\Psi| \neq 0$) begin
$j = \arg \max_{j, P_j \in \Psi} P_j$; \; // Choose operation with largest processing amount
$\Psi = \Psi - j, P_j$; \; // Remove chosen operation from set
for $k = 1, \ldots, W_j$ begin
$i = \arg \min_{i \in \Omega} L_i$; \; // Choose worker with smallest workload
$\Omega = \Omega - i, L_i$; \; // Remove chosen worker from set
$X_{i,j} = 1$; \; // Assign worker $i$ to operation $j$
$L_i = L_i + P_j$ \; // Increase workload of worker $i$
$F_i = C_j$; \; // Set new free time of worker $i$
end
$\Omega = \emptyset$; \; // Empty set of available workers
$t' = \min_{j, P_j > 0} E_j$ \; // Define time of next step
$t = t'$
end

18
Algorithm 4: CSGA(TM)
Initialisation: $t = 0$; $\text{pos}_{i,0} = \text{free}_i = \text{move}_i = 0 \forall i$; $z = 1$;

while ($t < T$) begin

$\Psi = \{ j \in J : E_j = t \}$; // Create list of operations

while $|\Psi| \neq 0$ begin // Assign workers to each operation in list

// Step 1: Assign workers to particular operations that ensure no movement from previous location.

Choose $j$ from front of list $\Psi$ and remove, i.e. $\Psi = \Psi - j$;

If $\exists i: \text{free}_i \leq t, \text{pos}_{i,z-1} = M_j$ then set $X_j = i; \text{pos}_{i,z} = M_j; \text{free}_i = C_j$;

else Insert $j$ into second list $\Omega$.

// Step 2: Assign workers to leftover operations randomly

while $|\Omega| \neq 0$ begin

Choose $j$ from front of list $\Omega$ and remove, i.e. $\Omega = \Omega - j$;

Select worker $i: \text{free}_i \leq t$

and set $X_j = i; \text{pos}_{i,z} = M_j; \text{free}_i = C_j$;

end

$E_j = \min_{j \in \Omega^*} E_j$ // Define time of next step

t' = t

$z = z + 1$;

end

Robert BURDETT is a researcher of the Operations Research Group in the School of Mathematical Sciences, Queensland University of Technology, Brisbane, Australia. His research interests include scheduling, mathematical modelling and heuristics. He has publications in the Journal of the Operational Research Society and in International Transactions in Operational Research.

Erhan KOZAN is an associate professor and head of the Operations Research Group in the School of Mathematical Sciences, Queensland University of Technology, Brisbane, Australia. He is a former national president of Operations Research Society for Australia. His research interests include scheduling, mathematical modelling and heuristics. He has published several papers in several operations research and transportation journals.