Decentralised Shared Resource Constraint Scheduling with Confidentiality Protection

Gaurav Singh[a,*], Christine M O'Keefe[b]

[a] *CSIRO Mathematics, Informatics and Statistics, Postal Bag 33, South Clayton VIC 3169, Australia*
[b] *CSIRO Mathematics, Informatics and Statistics, GPO Box 664, Canberra ACT 2601 Australia*

Abstract

As resources become scarce and expensive, it has become increasingly important for players in a decentralised supply chain to collaborate. One of the main challenges in collaboration is to find close to globally optimal solutions without sharing individual player's private data. Taking a decentralised resource constrained scheduling problem as an example we present a methodology which can be used to calculate correct lower and upper bounds on the objective functions without needing to share any private data from individual players.

Key words: decentralised scheduling, single shared resource, total weighted tardiness, release dates, precedences

1. Introduction

As resources like electricity and water are becoming very scarce and expensive, the supply chains are becoming very complicated and competitive. In order to reduce the cost of operating a supply chain it has also become important for various independent players within the supply chain to cooperate and share the scarce resources. In mineral supply chains these resources can be of many forms like electricity, number of trains, train tracks, stockyard capacities at the port and ship loading capacities. In what follows, the problem considered in this paper is motivated by the mining sites that are not connected to the main electricity grid. One power plant may provide electricity (the shared resource) to several mining operations. For environmental reasons the government may introduce a cap on the total amount of electricity that can be produced. Therefore, the mineral processing tasks of the mines have to be scheduled in such a way that the total electricity produced at the power plant does not exceed this cap. However, the mines are operated by competing companies and even though they are willing to cooperate, they do not wish to reveal confidential information about their operations to their competitors or to the power plant. Such confidential information would include details of the mineral processing tasks including: first time when the task could be started, amount of time required to process the job, due date and importance of completing the job before the due date. Such information could potentially be used by competitors or the power plant to attempt to prevent the mine delivering on time, or to attempt to gain a competitive advantage.

The problem considered in this paper is a particular form of resource constrained scheduling problem, and has attracted some recent attention [1, 2, 3, 4]. In this problem, processors (mines) have jobs (mining operations) to be executed that each require certain proportion of a shared resource (electricity). A limited quantity of this resource is provided by the central resource manager (the power plant). This corresponds to a *renewable resource* in the terms used by Brucker et al [5].

A comprehensive survey of models and solution methods for resource constrained scheduling can be found in [5, 6, 7]. Most of the problems in the literature consider minimising makespan as the objective. In this paper we consider minimising total weighted tardiness as the objective. Recently, there has also been some interest in minimising earliness-tardiness as the objective [8].

Shackelford and Corne [9] describe a collaborative evolutionary scheduling system, which enables the search to be guided by both the standard schedule quality criteria and also a human scheduler's non-formalised knowledge and experience.

An agent-based approach is presented by Lu and Yih [10] and Singh and Weiskircher [1, 2]. In [10], the software agents try to find balanced schedules to meet heterogeneous objectives of production entities within a collaborative manufacturing system in a manufacturing environment. The agents only exchange simple index data to reduce the communication and computation load of the control system. In [1, 2], a negotiation system was presented for solving a decentralised collaborative scheduling problem with a single shared resource and its performance was compared against a centralised genetic algorithm. It was shown that, even though the negotiation system obeys strict information decentralization to compute a feasible solution and while a central genetic algorithm has complete information and was given much longer computation time, the negotiation system can often find better solutions than the central genetic algorithm.

A more theoretical approach is taken in the paper by Cheng et al [11] where they look at the problem of multi-agent scheduling on a shared machine. They show that the problem is strongly NP-complete in general and present a pseudo-polynomial algorithm for integral weight as well as a fully

*Corresponding author
Email addresses: Gaurav.Singh@csiro.au (Gaurav Singh), Christine.OKeefe@csiro.au (Christine M O'Keefe)

polynomial-time approximation scheme.

Confessore, Giordani and Rismondo [12] suggested a multi-agent system that uses combinatorial auctions to allocate a shared resource among a number of projects. Their approach performs well on relatively small instances but is so far limited to exclusive shared resources that can only be used by one project at any point in time. For more papers on decentralised scheduling see [1, 13, 14, 15, 16] and [17].

Recently, Singh and Ernst [3] have presented a Volume algorithm [18] inspired Lagrangian relaxation based heuristic for the centralised version of the problem considered in this paper. That is, it was assumed that data is centralised and all the necessary information is available to all the players. It was shown that for large number of players, Lagrangian relaxation can out perform meta-heuristics like Simulated Annealing and Genetic Algorithms.

As the above literature review has shown there has been considerable interest in using distributed optimisation techniques in solving resource constrained scheduling problems. But, all the papers have assumed that the data is centralised and all the players are willing to share necessary information. The problem becomes more challenging when, for commercial sensitivity reasons, the players are not interested in sharing any information and in this case it is not easy to even calculate an objective value for a given schedule.

In this paper, we use secure multiparty computation [19, 20, 21] methods in the Lagrangian relaxation algorithm of [3] and show that it can be implemented without any player revealing any confidential information to any other processor or the central resource manager. These methods can be vulnerable to attack from colluding parties, however in our context we assume collusion does not occur as all parties are motivated to cooperate in order to ensure that the amount of electricity produced at the power plant does not exceed the cap. If this assumption is not valid, then additional measures as in Section 3 can be used to prevent collusion attacks for an honest majority of parties. We show how secure multiparty computation techniques can be applied to exactly calculate the global lower bounds and upper bounds on the problem without sharing any confidential information. We also present two different heuristic procedures which are also shown to converge to Nash's equilibrium. Results from computational experiments are also presented.

The rest of the paper is organized as follows. In the next Subsection we provide a formal description of the problem. In Section 2 we review the integer linear program for this problem from [3]. In Section 3 we present the details of Secure Multiparty Computation used in our algorithm. This is followed by a Lagrangian relaxation based approximation algorithm in Section 4. In Section 5 we present results from computational experiments and in Section6 we describe how our technique can be used for other problems. We then conclude in Section 7.

1.1. The Problem Description

The problem considered in this paper can be translated into the following machine scheduling problem. Suppose several jobs need to be scheduled on m processors, $\mathcal{M} = \{M_1, \ldots, M_m\}$, each one of them representing a mine/player. Each processor or mine or player M_i has a finite set $\mathcal{J}^i$ of n^i mineral processing jobs, $\{j^i_1, \ldots, j^i_{n^i}\}$, which need to be completed. Each job, $j \in \mathcal{J}^i$ is characterized by:

1. Release Time, r_j: the first time when the job becomes available.
2. Processing Time, p_j: the amount of time required in order to complete the processing of job j.
3. Due Date, d_j: the desired time by which the job needs to be completed.
4. Weight, w_j: a value, which reflects the importance of completing the job before its due date.
5. Resource Requirement of the Job, R_j: amount of the shared resource required in order to complete the job's processing.
6. Machine, M_i: the specified processor on which processing of the job needs to be completed.

There are no set-up times between the jobs. Each processor can process at most one job at a time. Each job can be processed only on its specified processor. No pre-emption is allowed. The processing of a job cannot be started before its release time. All the processors are available from time zero onwards. All data are assumed to be deterministic and integer valued, except for the w_j values which does not necessarily have to be an integer.

There are arbitrary precedence relations, $\rightarrow$, between jobs which belong to the same processor. In this relation, if $j, k \in \mathcal{J}^i$ and $j \rightarrow k$, then the processing of job k cannot be started until the processing of job j is completed on processor i.

We assume that the values r_j, p_j, d_j, w_j, R_j and M_j are confidential for every job $j \in \mathcal{J}^i, i = 1, \ldots, m$. We further assume that all the precedence relations $\rightarrow$ between jobs are confidential.

For every job $j \in \mathcal{J}^i$, a feasible schedule σ assigns a start time $s_j(\sigma)$ such that $s_j(\sigma) \geq r_j$. The completion time, $C_j(\sigma)$, of this job in schedule σ can be defined as, $s_j(\sigma) + p_j$. If $j, k \in \mathcal{J}^i$ and $j \rightarrow k$, then $C_j(\sigma) \leq s_k(\sigma)$.

Let σ be a feasible schedule and S_t be the set of all the jobs being processed or starting their processing in this schedule at time point t. More formally,

$$S_t = \{j | j \in \cup_i \mathcal{J}^i \ \& \ s_j(\sigma) \leq t < s_j(\sigma) + p_j\}$$

Also, let R_{max} be the maximum resource available at any given time. We will say σ is also a "resource-feasible" schedule if

$$\sum_{j \in S_t} R_j \leq R_{max} \quad \forall\, t \geq 0$$

That is the total resource consumed by all the jobs running or starting their processing at time t should not be more than the maximum available resource, R_{max}. For the purposes of this paper, we assume that R_{max} is known by all the processors.

The objective is to find a resource-feasible schedule σ, which minimizes the criterion of total weighted tardiness (TWT), which is defined as

$$TWT(\sigma) = \sum w_j \cdot T_j(\sigma),$$

where $T_j(\sigma)$ is the tardiness of the job j in schedule σ and is defined as $\max\{C_j(\sigma) - d_j, 0\}$.

Singh and Ernst [3] showed that, the problem for even a single mine, denoted as $1|d_j, r_j, prec|\Sigma w_j \cdot T_j$, is also NP-hard. Therefore, they proposed a method designed to find near-optimal solutions, in the case that there is free exchange of required information between the processors.

In this paper, we modify the Singh-Ernst [3] Lagrangian relaxation based algorithm to adapt it to the case in which the processors do not reveal their confidential information r_j, p_j, d_j, w_j, R_j and M_j. However, we do assume that the value R_{max} is known by each processor.

2. An Integer Linear Program

Singh and Ernst [3] have presented a integer linear programming formulation of the centralised problem described in previous section. For sake of completeness we present the model here. Let $\mathbb{J} = \cup_{i=1}^{m} \mathcal{J}^i$ be the set of all the jobs. For every job $j \in \mathbb{J}$ we first calculate, c_{jt}, which reflects the cost of starting j at time $t - p_j$ and finishing at t. More formally,

$$c_{jt} = w_j \max\{t - d_j, 0\} \; \forall t, \;\; \forall j \in \mathbb{J}$$

Note that each value c_{jt} depends on the confidential information w_j and d_j of job $j \in \mathbb{J}$. It can be calculated by the processor M_i on which job j will be processed, and can be held confidential by that processor.

We also define following binary variables

$$z_{jt} = \begin{cases} 1 & \text{if } j \in \mathbb{J} \text{ is completed by time } t \\ 0 & \text{otherwise} \end{cases} \quad \forall t, \;\; \forall j \in \mathbb{J}$$

The objective function to be minimised is the total weighted tardiness, given as

$$\min \sum_{j \in \mathbb{J}} \sum_{t>0} c_{jt}(z_{jt} - z_{jt-1}) \tag{1}$$

It can be noted that the total weighted tardiness is a function of processor's confidential information. Our revised algorithm will avoid the need for the processors to reveal confidential information in minimising this objective function.

The various constraints can also be modeled as

1. *Release date constraints:*

$$z_{jt} = 0 \; \forall j \in \mathbb{J}, \;\; \forall t = 0, \ldots, r_j + p_j - 1$$

2. *Not more than one job on any processor:*

$$\sum_{j \in \mathcal{J}^i} (z_{jt+p_j} - z_{jt}) \leq 1 \; \forall t, \; \forall i = 1, \ldots, m$$

3. *Once job is completed it stays as completed:*

$$z_{jt} \geq z_{jt-1} \; \forall j \in \mathbb{J}, \;\; \forall t > 0$$

4. *All jobs must be completed:*

$$z_{jT} = 1 \;\; \forall j \in \mathbb{J}$$

where T is the end of time horizon.

5. *Precedence Relations:*

$$z_{kt} \leq z_{jt-p_k} \; \forall j \to k, \;\; \forall t$$

6. *Central Resource Constraint:*

$$\sum_{j \in \mathbb{J}} R_j(z_{jt+p_j} - z_{jt}) \leq R_{max} \; \forall t. \tag{2}$$

Note that in this formulation, constraints 1-5 depend only on local processor information. In other words, if each processor checks these constraints for its own jobs then the constraints are checked for every job. Each processor can determine whether the relevant constraints are satisfied without needing information from any other processor and without revealing any information to any other processor. In contrast, constraint 6 requires information on all jobs, that is, requires information from all processors. Our revised algorithm will enable this constraint to be checked without the need for any processor to reveal confidential information to any other party.

Our revised algorithm depends on Secure Multiparty Computation, as introduced in the next Section.

3. Secure Multiparty Computation

In this Section we give a brief introduction to *secure multiparty computation,* including details about *secure sum,* the technique used in our heuristic [19], see [20, 21].

Suppose that a number of participants wish to each provide an input and jointly conduct a computation such as finding the sum of their individual inputs. Suppose further that the input held by each participant is confidential and so the participants do not wish to reveal their input to the others. If there is an additional independent party trusted by all the participants, a *trusted third party*, then all participants can send their individual inputs to the trusted third party who then conducts the computation and returns the result to the participants. In this way, at the end of the computation, no participant knows anything except its own input and the result, however the trusted third party knows all inputs as well as the result. The idea of secure multiparty computation is to avoid the need for a trusted third party, while still ensuring that at the end of the computation, no participant knows anything except its own input and the result.

3.1. Secure Sum

In the context of this paper, we only require *secure sum*; and we follow the account in [20]. Suppose that sites $1, 2, \ldots, s$ hold values $v_1, v_2, \ldots, v_s$ respectively, and know that the sum $V = \sum_{i=1}^{s} v_i$ lies in the interval $[0, n-1]$. One site is designated to be the *master* site, say site 1. Site 1 generates a random number g, uniformly chosen on the interval $[0, n-1]$, adds it to its local value v_1 and sends the sum $V_1 = g + v_1 \mod n$ to site 2. Since g was chosen uniformly from $[0, n-1]$, the number $g + v_1 \mod n$ is also distributed randomly on $[0, n]$ and site 2 learns nothing about the value of v_1. For $k = 2, \ldots, s-1$, the following steps are performed: site k receives the value $V_{k-1} = g + \sum_{i=1}^{k-1} v_i \mod n$. This value is uniformly distributed on $[0, n-1]$, so site k learns nothing. Site k now computes $V_k = V_{k-1} + v_k \mod n$ and passes it to site $k+1$. Site

s performs the calculation as above, and sends the result to site 1. Site 1 now subtracts $g \bmod n$ to obtain $V_s = V = \sum_{i=1}^{s} v_i$.

As discussed in [20], this method is vulnerable to attack if parties collude. For example, sites $k-1$ and $k+1$ can determine the value v_k. The method can easily be extended to work for an honest majority as follows. First, each site k divides its value v_k into ℓ *shares* so that the sum of the shares is the value v_k. The secure sum protocol is repeated ℓ times, with each site using one of its shares in each iteration, but with the sites numbered in a different order each time so that no site has the same neighbor twice. Then the ℓ results are added to obtain the overall sum. Increasing ℓ increases the number of parties that would need to collude to learn information about a value.

4. A Lagrangian Relaxation based Algorithm

In this Section we present a modified version of the Lagrangian relaxation based method from [3] which was inspired by the Volume Algorithm presented in [18]. We relax the set of constraints (2), the resource consumption constraints using Lagrangian relaxation techniques and introduce the Lagrange multipliers, λ_t, for time point t. The motivation to use Lagrangian relaxation techniques was the fact that it allowed to decompose the problem into subproblems for each processors, which can then be solved independently.

By relaxing the constraints (2), the Lagrangian Dual can be obtained as:

$$LRR(\lambda) = \min \sum_{j \in \mathbb{J}} \sum_{t>0} c_{jt}(z_{jt} - z_{jt-1}) + \sum_{t} \lambda_t \left(\sum_{j \in \mathbb{J}} R_j((z_{jt+p_j} - z_{jt}) - R_{max}) \right) \quad (3)$$

The above equation can be rearranged to give

$$LRR(\lambda) = \min \sum_{j \in \mathbb{J}} \sum_{t>0} c'_{jt}(z_{jt} - z_{jt-1}) - R_{max} \sum_{t} \lambda_t,$$

where

$$c'_{jt} = c_{jt} + \sum_{i=1}^{\min\{p_j, t\}} \lambda_{t-i} R_j. \quad (4)$$

One can note that, c'_{jt} gives the total "cost" of completing the job j at time t, as it is sum of job's weighted tardiness and the cost of using R_j amount of resource during the job's execution. The objective for the i^{th} subproblem then becomes

$$LRR(\lambda)^i = \min \sum_{j \in \mathcal{J}^i} \sum_{t>0} c'_{jt}(z_{jt} - z_{jt-1}).$$

Note that the i^{th} subproblem only involves information about the processor M_i through the variables c_{jt} and R_j; it does not require information about any other processor.

Algorithm 1 provides the various steps involved in this algorithm. This algorithm is motivated by the volume algorithm [18] and incorporates in each iteration the direction of movement as a convex combination of the current and previous subgradients. The main idea is to accelerate solution's convergence by avoiding the usual zig-zag behavior of the subgradient algorithm. Here, at each iteration k, vectors $D^k = \{D_1^k, \ldots, D_T^k\}$ represent the violation of constraints (2) at each time t, S represents the convex combination of the current and previous subgradients and β provides the direction of the current subgradient.

The algorithm proceeds in two stages. In the first stage, the individual processor independently solve the subproblems using only their local information. So the first stage does not require any processor to reveal confidential information to another processor or to any other party. The second stage of the algorithm, which is a collaborative protocol to seek an overall solution, makes use of secure multiparty computation to avoid the need for any processor to reveal confidential information to another processor or to any other party.

4.1. An Iterative Heuristic

As Singh-Ernst [3] noted when all the sub-problems comprise of only one job, with $r_j = 0, w_j = 1$ and $R_j = R_{max}$ for all the jobs, $j \in \mathbb{J}$. Then, finding a "resource-feasible" schedule is equivalent to $1||\sum T$ problem, which is known to be NP-hard [22]. Therefore, in general sense, the problem of finding a "resource-feasible" schedule from a feasible schedule is NP-hard. Which justifies the use of a heuristic procedure to calculate an upper bound for the problem.

In this section we present a general framework that is used to calculate an upper bound for our problem. In this framework the processors apply strategies using information from the system in order to achieve their goals. The processors, however, have incomplete information of the full system as the data is decentralised.

In each step of the framework the processors schedule their jobs based on release dates, precedence constraints, due dates and job weights and taking into account the availability of the shared resource in the complete schedule. The processors use ILP from Section 2 with constraints relevant to only their requirements and instead of constraint (2) the following constraint is used

$$\text{if } R_j > RR_t, \text{ then } z_{jt+p_j} \leq z_{jt}, \ \forall t \ \ \forall j \in \mathcal{J}^i$$

where RR_t is the available resource at time t. In the first iteration it is assumed that the all of the resource is available i.e. $RR_t = R_{max}$. The constraint ensures that a processor's job is not running during the time t if the amount of resource available is not sufficient to complete this job.

The resource manager receives a schedule, σ_i from processor i, and resource requirement, R_j, for every job $j \in \mathcal{J}^i$. The resource manager internally sets

1. release date of a job j as $s_j(\sigma_i) \ \ \forall j \in \mathbb{J}$, and
2. if $s_j(\sigma_i) < s_k(\sigma_i)$ then sets $j \rightarrow k \ \ \forall j, k \in \mathcal{J}^i$.
3. processing time p_j of a job can also be calculated from $s_j(\sigma_i)$ and resource utilisation.

The resource manager next creates a random list of jobs and Algorithm 2 is used to find a "resource-feasible" schedule, σ_c.

Algorithm 1: Lagrangian Relaxation based Algorithm

Data: Initialization. Set $k = 0$, $\rho = 1.5$, $LB^* = 0$, $UB^* = \infty$, $\lambda^0 = (\lambda_1^0, \lambda_2^0, \cdots, \lambda_T^0) = (0, 0, \cdots, 0)$, $S = (0, \cdots, 0)$, $\lambda^* = \lambda^0$, $\gamma_{max} = 0.65$, $gap = \infty$

while $k < 500 || \ \rho \geq 0.0001 || \ gap \ \geq \ 0.05$ **do**

The processors independently solve the subproblems with $LRR(\lambda^k)^i$ as the objective for the i^{th} subproblem, without the receipt or transmission of any confidential information. Let $\{z'_{jt}\}\ \forall t, j \in \mathbb{J}$ be the solution from the subproblems

Find the resource remaining,

$$D_t^k = R_{max} - \sum_{i=1}^{m} \sum_{j \in \mathcal{J}^i} (z'_{jt+p_j} - z'_{jt}) \times R_j, \forall t$$

Note that the processors can independently compute the subtotals $\sum_{j \in \mathcal{J}^i} (z'_{jt+p_j} - z'_{jt}) \times R_j$, and then can use the Secure Sum protocol of Section 3.1 to compute the overall sum without revealing their individual subtotals to each other

Set the lower bound,

$$LB^k = \sum_i LRR(\lambda)^i - R_{max} \sum_t \lambda_t^k.$$

The processors use the Secure Sum protocol of Section 3.1 to compute $\sum_i LRR(\lambda)^i$ without revealing their individual subtotals to each other

Use heuristic procedure described in Section 4.1 or in Section 4.2 to calculate an upper bound, UB^k.

if $UB^* > UB^k$ **then**

Set $UB^* = UB^k$

Set $\beta = S \cdot D^k$ and $||S||^2 = S \cdot S$

if $\beta \geq 0$ & $LB^* < LB^k$ **then**

$\lambda^* = \lambda^k$

$LB^* = LB^k$

$\rho = \min\{1.3\rho, 2.0\}$

else

$\rho = 0.95\rho$

Let γ^* be the solution that minimises $||\gamma D^k + (1 - \gamma)S||$.

if $\gamma^* < 0$ **then**

$\gamma = \gamma_{max}/10.0$

else

$\gamma = \min\{\gamma^*, \gamma_{max}\}$

Set $S = \gamma S + (1 - \gamma)D^k$

The Lagrange multipliers $(\forall t)$ are adjusted as

$$\lambda_t^{k+1} = \max\left\{0, \ \lambda_t^k - \frac{\rho(UB^* - LB^*)}{||S||^2} S_t\right\}$$

Set $gap = \frac{UB^* - LB^*}{LB^*}$ and $k = k + 1$

Finally, the resource manager uses Secure Sum described in Section3.1 to find the total weighted tardiness of the schedule σ_c. Theorem 1 shows that the schedule σ_c is a resource-feasible schedule.

If for every job $j \in \mathbb{J}$, $s_j(\sigma_i) = s_j(\sigma_c)$, then the process stops. As this indicates that the schedules sent by the processors had no resource conflict. Otherwise, the resource manager returns back a modified schedule of each processor together with the resource profile $\{RR_t\}$, which indicates the remaining shared resource at any time t. Each processor then tries to improve on the schedule sent by the resource manager by rearranging its jobs and taking the resource profile into account. The iteration continues until no processor can find a schedule which can improve the schedule sent by the resource manager. There are two things which can be noted about this procedure:

- The final solution where the solution terminates is a Nash's equilibrium, as no processor is able to improve their own schedule by keeping every other processor's resource allocations as is.
- Also, as there are only finite set of schedules, the procedure must terminate after certain iterations.

Theorem 1. *The schedule generated by the resource manager is a resource-feasible schedule for all processors.*

PROOF. Let σ_c be the schedule generated by the resource manager at some iteration of the procedure described in Section 4.1 and let σ_i be the ith processor's feasible solution which was used to generate the schedule σ_c. It is easy to see from Algorithm 2, that σ_c always satisfies the resource constraint. It therefore, suffices to show that the σ_c is feasible for an individual processor. In order to do so there are two important properties to be checked. Firstly, a job j must not start before its release date, r_j. Since, $s_j(\sigma_c) \geq s_j(\sigma_i)$ and $s_j(\sigma_i) \geq r_j$, it shows that $s_j(\sigma_c) \geq r_j$.

Secondly, if jobs $j \rightarrow k$ for a processor i, then as σ_i is a feasible schedule $s_j(\sigma_i) < s_k(\sigma_i)$. Therefore, using condition 2, $j \rightarrow k$ is preserved in the schedule σ_c as well.

Algorithm 2: A Serial List Scheduling based Algorithm

Data: A list of jobs $\mathcal{J} = \{j_1 \ldots j_n\}$

Result: A "resource-feasible" schedule σ,.

while $\mathcal{J} \ != \ \emptyset$ **do**

Let j be the first job such that for no $i \in \mathcal{J}, i \rightarrow j$

set $t = \max\{r_j, \max_{i:i \rightarrow j}\{s_i(\sigma) + p_i\}\}$

Let $t_m \geq t$ be the earliest time when processor M_j is idle and at least R_j amount of resource is available during the interval $[t_m, t_m + p_j)$

set $t \ = \ t_m$ and $s_j(\sigma) \ = \ t$

$\mathcal{J} \ = \ \mathcal{J} \setminus \{j\}$

return σ

It is easy to see that if $\sigma*$ is an optimal schedule for the problem, then it is possible to construct a list of all the jobs, for which Algorithm 2 produces the same schedule as $\sigma*$.

4.2. A leader-follower based heuristic

At each iteration of this heuristic the resource manager randomly picks a processor with unscheduled jobs as leader. The leader then uses the ILP to optimise only its objective within the current availability of the resource. Based on the schedule of the leader the resource profile is updated and another processor with unscheduled jobs is selected as leader. The algorithm continues until jobs of all the processors have been scheduled. Once again, the resource manager uses Secure Sum described in Section 3.1 to find the total weighted tardiness of the final schedule obtained by this procedure. It is known ([23]) that the procedure terminates with a solution which is also a Nash's equilibrium. Algorithm 3 provides a high-level description of this algorithm.

Algorithm 3: Leader-Follower based Heuristic

Input: Set. $b = 0, converged = false, UB^* = \infty$

while $b < 100$ || $converged = false$ **do**
  Create a random list of processors
  Use the leader-follower procedure to create a resource-feasible schedule
  Use the secure sum procedure to calculate UB^b
  if $UB^* > UB^b$ **then**
    $UB^* = UB^b$
  if *no processor is able to reduce its TWT* **then**
    set $converged = true$
  else
    $b+ = 1$
return UB^*

5. Computational Experiments

For a fair comparison we used the same instances as in [3]. There were 800 problem instances with 3 to 12 processors each with 10.5 jobs on average. This gave us on average 80 instances for a particular number of processor. The instances were also generated such that at least two processors can execute jobs simultaneously. In our experiments, we applied four solution methods to the problem instances:

1. **LR:** The Lagrangian Relaxation based algorithm presented in [3].
2. **LRIH:** The Lagrangian Relaxation method described in 4 and upper bound procedure as "iterative heuristic" described in 4.1.
3. **LRLF:** The Lagrangian Relaxation method described in 4 and upper bound procedure as "leader-follower heuristic" described in 4.2. As every iteration of this heuristic is cpu-intensive, the algorithm is only called every 10th iteration and every iteration when the best solution is updated.
4. **SAS:** The simulated annealing based heuristic presented in [3].

Out of these four methods, LRIH and LRLF methods are decentralised methods that use the secure computation technique and other two methods are centralised methods from the literature used for comparison purposes.

All algorithms ran on 64-bit 2.93 GHz Intel Xeon(R) with 8 GB of main memory. We measured two values for each problem instance for each of the three methods: solution quality i.e. Total Weighted Tardiness (TWT) and the CPU time in secs. Note that for both measures, smaller values are better. When comparing any two methods, say A and B, we first calculated the ratios $TWT(A)/LB$ and $TWT(B)/LB$, where $TWT(A)$ is the Total Weighted Tardiness found by the method A and LB is the lower bound found by the LR method. We then computed for each problem instance the difference between these ratios for the two methods. Finally, Student's t-test was applied to the resulting data in order to obtain the estimated mean and the 95% confidence interval for the mean. This allowed us to filter out the absolute differences between the objective function values of different problem instances, since we are only interested in which algorithm is better for each instance. For all two measures, we divided the problem instances into 10 groups corresponding to the number of processors.

Figure 1 shows the 95% confidence intervals of Student's t-test on the differences of relative TWT between pairs (LRIH, LR), (LRLF, LR), (LRIH, SAS), (LRLF, SAS) and (LRLF,LRIH). The results indicate following:

1. For 3,4,5 processors the difference between the two heuristic procedure was only marginal.
2. For 6 or more processors the LRIH method found better results than the LRLF method. This difference does not increases much with increase on number of processors.
3. For up to 7 processors, the SAS method found better solutions than the LRLF or LRIH methods. But for larger number of processors it is interesting that the LRIH and LRLF methods found better solutions than the SAS method. There is also a very clear pattern that for large number of processors the difference between quality of solutions found by LRIH and LRLF methods is only increasing.
4. For small number of processors (3,4 and 5) the LRIH and LRLF methods could find better solutions than the LR method. But for larger number of processors the LR method found better solutions. Although, this difference is not increasing much with increase in number of processors.
5. The difference between LRIH method and LR method for for more than 6 processors was close to 5%. We find it interesting that the LRIH method has incomplete information of the problem as compared to LR method and still it manages to find good solutions. This also reflects on the price of anarchy for these problems as around 5%.

Figure 2 shows the 95% confidence intervals of Student's t-test on the difference of relative lower bounds found by pairs (LRIH,LR) and (LRLF,LR). As the results indicate the quality

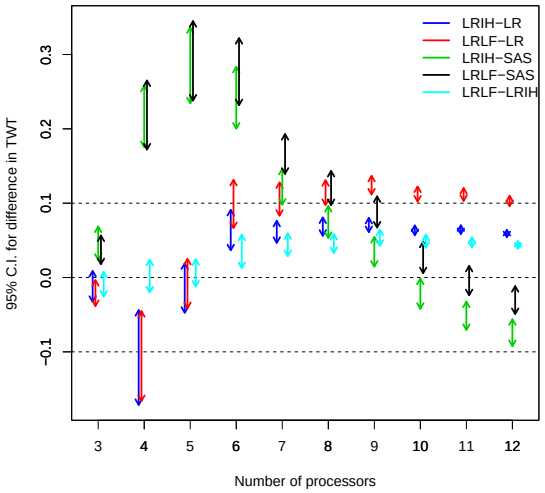

Figure 1: Confidence intervals of Student's t-test on the differences of the relative TWT

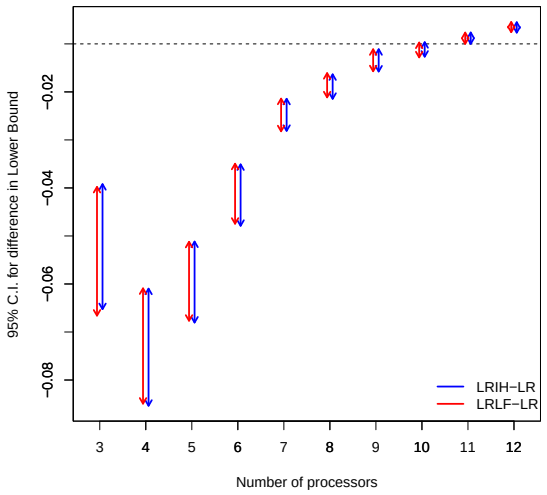

Figure 2: Confidence intervals of Student's t-test on the differences of the lower bound

of lower bound found by the LRLF and LRIH methods do not differ much and for small processors number the lower bounds are worst than the LR method. But, for instances with large number of processors the difference is very marginal.

Figure 3 shows the average CPU time in seconds for various methods. As this figure indicates that method LRIH is quite comparable with LR method and on average used same (and sometimes smaller) time compared with LR method. For large number of processors LRLF method has a high cpu utilisation. This is because, with increase in the number of processors, the Leader-Follower heuristic takes longer time to converge to an equilibrium.

6. A Generalised Procedure

The Lagrangian relaxation method presented in this paper can be easily extended to other problems which have independent players and data is decentralised. Indeed, these problems will usually have some constraints which are local to the particular player and then there are some constraints which link these independent players like the resource constraint in the problem considered here. In this case, as shown in this paper, all the linking constraints can be relaxed and a Lagrangian relaxation based algorithm can be developed. Using secure sum techniques, the algorithm can estimate the true lower and upper bounds without disclosing any private information of individual player.

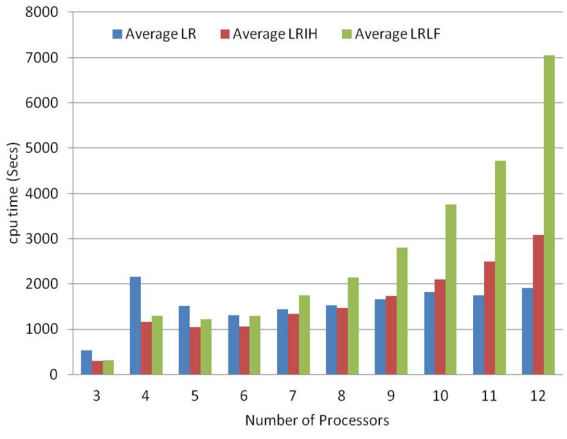

Figure 3: Average CPU time (secs) for methods LR, LRIH and LRLF

7. Conclusion

Motivated from decentralised supply chains, in this paper, we presented a decentralised Lagrangian relaxation method, which allows us to calculate the true lower and upper bounds on the problem without disclosing any private information of

any player. We also present two heuristic procedures which converge to Nash's equilibrium and compare their performance with algorithms in the literature. Our results show that even with partial information the algorithms presented in this paper are quite competitive and find solutions very close to the centralised algorithms in the literature.

The algorithm presented in this paper cannot be applied to the case when the objective function is non-additive, for example minimising maximum tardiness. In the future, we want to develop strategies which can be used for the case when the objective function is non-additive.

References

[1] G. Singh, R. Weiskircher, Collaborative resource constraint scheduling with a fractional shared resource, in: proceedings of 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-08), IEEE Computer Society, Sydney, Australia, 2008, pp. 359–365.

[2] G. Singh, R. Weiskircher, A multi-agent system for decentralised fractional shared resource constraint scheduling, Web Intelligence and Agent Systems 9 (2) (2011) 99–108.

[3] G. Singh, A. Ernst, Resource constraint scheduling with a fractional shared resource, Operations Research Letters 39 (5) (2011) 363–368.

[4] D. Thiruvady, A. Ernst, G. Singh, Parallel ant colony optimization for resource constrained multiple machine job scheduling, Computers and Operations Research submitted.

[5] P. Brucker, A. Drexl, R. Mohring, K. Neumann, E. Pesch, Resource-constrained project scheduling: Notation, classification, models, and methods, European Journal of Operational Research 112 (1) (1999) 3–41.

[6] E. Demeulemeester, W. Herroelen, Project Scheduling: A Research Handbook, Kluwer: Boston, 2002.

[7] C. S. K. Neumann, J. Zimmermann, Project Scheduling with Time Windows and Scarce Resources, Springer: Berlin, 2003.

[8] F. Ballestin, N. Trautmann, An iterated-local-search heuristic for the resource-constrained weighted earliness-tardiness project scheduling problem, International Journal of Production Research 46 (22) (2008) 6231–6249.

[9] M. Shackelford, D. Corne, Collaborative evolutionary multi-project resource scheduling, in: Proc. of 2001 Congress on Evolutionary Computation, Vol. 2, 2001.

[10] T. P. Lu, Y. Yih, An agent-based production control framework for multiple-line collaborative manufacturing, International Journal of Production Research 39 (10) (2001) 2155–2176.

[11] T. C. E. Chenga, C. T. Ng, J. J. Yuan, Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs, Theoretical Computer Science 362 (2006) 273–281.

[12] G. Confessore, S. Giordani, S. Rismondo, A market-based multi-agent system model for decentralized multi-project scheduling, Annals of Operations Research 150 (1) (2006) 115–135.

[13] E. Kutanoglu, D. Wu, On combinatorial auction and lagrangean relaxation distributed resource scheduling, IIE Transactions 31 (1999) 813–826.

[14] X. Li, L.-K. Soh, Hybrid negotiation for resource coordination in multiagent systems, Web Intelligence and Agent Systems 3 (4) (2005) 231–259.

[15] S. Albayrak, D. Milosevic, Multi-domain strategy coordination approach for optimal resource usage in agent based filtering framework, Web Intelligence and Agent Systems 4 (2) (2006) 239–253.

[16] P. Dewan, S. Joshi, Dynamic single-machine scheduling under distributed decision-making, International Journal of Production Research 38 (16) (2000) 3759–3777.

[17] P. Dewan, S. Joshi, Auction-based distributed scheduling in dynamic job shop environment, International Journal of Production Research 40 (5) (2002) 1173–1191.

[18] F. Barahona, R. Anbil, The volume algorithm: producing primal solutions with a subgradient method, Mathematical Programming 87 (3) (2000) 385399.

[19] A. Yao, How to generate and exchange secrets, in: Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, IEEE, 1986, pp. 162–167.

[20] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, M. Zhu:2002, Tools for privacy preserving distributed data mining, SIGKDD Explorations 4 (2) (2002) 28–34.

[21] B. Pinkas, Cryptographic techniques for privacy-preserving data mining, SIGKDD Explorations 4 (2) (2002) 12–19.

[22] J. K. Lenstra, A. H. G. R. Kan, P. Brucker, Complexity of machine scheduling problems, Annals of Discrete Mathematics 1 (1977) 343–362.

[23] T. Roughgarden, Stackelberg scheduling strategies, in: 31st ACM Symposium on Theory of Computing (STOC), ACM, New York, USA, 2001, pp. 104–113.