SCHEDULING PROJECTS WITH LABOR CONSTRAINTS

C. C. B. Cavalcante [1], C. Carvalho de Souza[1], M.W.P. Savelsbergh[2], Y. Wang[3] and L.A. Wolsey[4]

July 20, 1999

Abstract

In this paper we consider a labor constrained scheduling problem (LCSP) which is a simplification of a practical problem arising in industry. Jobs are subject to precedence constraints and have specified processing times. Moreover, for each job the labor requirement varies as the job is processed. Given the amount of labor available in each period, the problem is to finish all the jobs as soon as possible, that is, to minimize the *makespan*, subject to the precedence and labor constraints. Several Integer Programming (IP) formulations for this problem are discussed and valid inequalities for these different models are introduced. It turns out that a major drawback in using an IP approach is the weakness of the lower bound relaxations. However, we report computational experiments showing how the solution of the linear relaxation of the IP models can be used to provide good schedules. Solutions arising from these LP-based heuristics are considerably improved by local search procedures. We further exploit the capabilities of local search for LCSP by designing a Tabu search algorithm. The computational experiments on a benchmark data set show that the Tabu search algorithm generates the best known upper bounds for almost all these instances. We also show how IP can be used to provide reasonably good lower bounds for LCSP when the makespan is replaced by suitably modified objective functions. Finally some directions for further investigations which may turn IP techniques into a more interesting tool for solving such a problem are suggested.

Keywords: Project Scheduling, Labor Constraints, Integer Programming, Valid Inequalities, Tabu Search, LP-based Ordering Heuristics

[1]Instituto de Computação, Universidade Estadual de Campinas — UNICAMP, Caixa Postal 6176 – CEP: 13083-970 – Campinas, SP – Brazil. This research was supported by FAPESP (grants 97/02990-3 and 96/10270-8), by CNPq (grant number 300883/94-3) and PRONEX 107/97 (MCT/FINEP).

[2]School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0205, USA. This research was supported by NFS grant DMI-9700285.

[3]Navigation Technologies, 10400 W. Higgins Road, Rosemont, IL 60018. Work carried out while holding a research fellowship at CORE.

[4]CORE and INMA, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

1 Introduction

The labor constrained scheduling problem (LCSP) involves sequencing a set of jobs subject to precedence constraints. Each job has a specified processing time, and a labor profile which typically varies as the job is processed. Given the amount of labor available in each period, the problem is to finish all the jobs as soon as possible (minimize *makespan*, subject to the precedence and labor constraints). This problem is not new, see for instance Wagner [28]. The particular problem motivating this study appears in Heipcke [15] and is a simplification of an industrial problem from BASF. Heipcke [15] develops a basic constrained programming approach to construct feasible schedules. This approach is extended in Heipcke and Colombani [17] and used to solve several small instances to optimality. However, the quality of the schedules produced appears to degrade significantly when the instances get larger.

The limited success of the constrained programming approach prompted us to investigate the viability of other solution approaches. We report on some of these efforts in this paper. LCSP is a real challenge to the integer programming community as well, because, in spite of a significant research effort, little or no progress has been made in finding nontrivial lower bounds, and thus in measuring the quality of the solutions for hard highly constrained instances. On the positive side, LP-based ordering heuristics and local search methods lead to feasible schedules, and the quality of the schedules found can be compared.

We now outline the contents of this paper. In Section 2 we report on our efforts to model LCSP as an integer program. Different time-indexed formulations are presented, including a block formulation specially designed to treat the BASF instances which involve scheduling blocks of identical jobs. In Section 3 we examine LP-based ordering heuristics. As the name suggests, LP-based ordering heuristics convert an ordering obtained from the solution to the LP relaxation of some IP formulation of the problem into a feasible schedule. LP-based ordering heuristics for the LCSP, when combined with local search, produce high quality schedules. Unfortunately, the approach is limited to medium sized instances because of the difficulty of solving the LP relaxations directly. In Section 4, we report on other attempts to use the integer programming formulations to produce lower bounds and/or feasible schedules. In particular, we discuss some additional valid inequalities and some attempts to solve small instances to optimality. More specifically, we present a branch-and-prune approach, which uses a modified objective function and a modified tree search, and appears to improve on a direct branch-and-bound approach. The LP-based ordering heuristics of Section 3 produce better solutions when followed by simple local improvements procedures. This suggests that local search algorithms, even without a good initial schedule provided by the LP-based ordering heuristics, may be able to produce good solutions for LCSP. In Section 5 we discuss a tabu search algorithm that handles much larger instances (since it does not require the solution of a large LP

to get an initial schedule), and that produces high quality solutions. Two common features in Sections 3 - 5 are the use of modified objective functions, as IP is traditionally ineffective in handling makespan, or more generally min max objective functions, and the use of a "reverse" instance whose makespan is identical. Finally Section 6 contains some brief conclusions and suggestions for further work.

2 Formulations

An instance of LCSP is determined by a set $N = \{1, \ldots, n\}$ of jobs, the processing time p_j and labor profile $(\ell_{j,1}, \ldots, \ell_{j,p_j})$ of each job $j \in N$, and a digraph $D = (N, A)$ representing the precedence constraints between jobs. An amount L of labor is available in each period. The objective is to find a nonpreemptive schedule that minimizes the makespan. It is assumed that the parameters L, p_j, $\ell_{j,i}$ and T are all integer-valued.

Because of the varying labor profiles, it appears impossible to avoid the use of time-indexed formulations [29]. Such formulations have been proposed many times [19, 20], but little computational experience has been reported except for some recent work on single machine scheduling [26, 2] and a successful application to an air traffic control problem [4]. Thus we choose a sufficiently long time horizon T and introduce periods $1, 2, \ldots, T$ representing the intervals $[0, 1), [1, 2), \ldots, [T-1, T)$.

2.1 The job formulation

Time-indexed formulations are typically based on two closely related sets of variables, namely $x_{j,t} = 1$ if job j starts in period t, and 0 otherwise, or $z_{j,t} = 1$ if job j starts in or before period t, and 0 otherwise. If job j can only start between periods $e(j)$ and $f(j)$, we have the following relationship between the x and z variables: $x_{j,e(j)} = z_{j,e(j)}$ and $x_{j,t} = z_{j,t} - z_{j,t-1}$ for $t = e(j)+1, \ldots, f(j)$. For simplicity, we add to the problem a final dummy job that can only start when all the real jobs have terminated. Thus N and D are modified, and the new objective is to minimize the start time of the last dummy job $n \in N$. We take $e(n) = \max_{j<n}\{e(j) + p_j\}$, $f(n) = T$, $p(n) = T - e(n) + 1$ and $\ell_{n,u} = L$ for all u.

We now give two IP formulations which we express in both the x and z variables. Also we introduce the variables s_j for $j \in N$ to denote the period in which job j starts.

The weak x-formulation

$$\min s_n \tag{1}$$

$$\sum_{t=e(j)}^{f(j)} x_{j,t} = 1 \text{ for } j \in N \tag{2}$$

$$s_j = \sum_{t=e(j)}^{f(j)} t x_{j,t} \text{ for } j \in N \quad (3)$$

$$s_j \geq s_i + p_i \text{ for } (i,j) \in A \quad (4)$$

$$\sum_{j=1}^{n} \sum_{u=1}^{p_j} \ell_{j,u} x_{j,t-u+1} \leq L \text{ for } t = 1, \ldots, T \quad (5)$$

$$x_{j,t} \in \{0,1\} \text{ for } t = e(j), \ldots, f(j),\ j \in N \quad (6)$$

The weak z-formulation

$$\min s_n \quad (7)$$

$$z_{j,f(j)} = 1 \text{ for } j \in N \quad (8)$$

$$z_{j,t} \leq z_{j,t+1} \text{ for } t = e(j), \ldots, f(j)-1,\ j \in N \quad (9)$$

$$s_j = f(j) + 1 - \sum_{t=e(j)}^{f(j)} z_{j,t} \text{ for } j \in N \quad (10)$$

$$s_j \geq s_i + p_i \text{ for } (i,j) \in A \quad (11)$$

$$\sum_{j=1}^{n} \sum_{u=1}^{p_j} \ell_{j,u} (z_{j,t-u+1} - z_{j,t-u}) \leq L \text{ for } t = 1, \ldots, T \quad (12)$$

$$z_{j,t} \in \{0,1\} \text{ for } t = e(j), \ldots, f(j),\ j \in N \quad (13)$$

Constraints (2) or (8), (9) impose that each job j is carried out, and (3) and (10) express s_j in terms of the $x_{j,t}$ and $z_{j,t}$ variables respectively. Constraints (4) or (11) represent the precedence constraints. Finally (5) and (12) are the labor constraints.

Now we present two stronger formulations in which the precedence constraints are tightened, see [29], and the start variables s_j are eliminated.

The strong x-formulation

$$\min \sum_{t=e(n)}^{T} t x_{n,t} \quad (14)$$

$$\text{s.t. } (2), (5), (6)$$

$$\sum_{u=e(i)}^{t} x_{i,u} \geq \sum_{u=e(j)}^{t+p(i)} x_{j,u} \text{ for } (i,j) \in A,\ t = e(i), \ldots, f(i)-1. \quad (15)$$

The strong z-formulation

$$\max \sum_{t=e(n)}^{T} z_{n,t} \tag{16}$$

$$\text{s.t. } (8), (9), (12), (13)$$

$$z_{i,t} \geq z_{j,t+p_i} \text{ for } (i,j) \in A,\ t = e(i), \ldots, f(i) - 1. \tag{17}$$

Note that a priori the weak x-formulation appears preferable to the weak z-formulation because the latter has the $O(nT)$ constraints (9). On the other hand, ignoring the labor constraints (5) or (12), the constraints of the strong z-formulation are much simpler than those of the strong x-formulation. In particular we know from the following result that the problem (16), (17), (8), (9), (13), without the labor constraints, is easily solved.

Proposition 1 *The matrix consisting of the constraints (8), (9), (17) is totally unimodular.*

As observed in [14], this result follows immediately since each constraint contains at most two nonzero coefficients, and the coefficients are +1,-1 respectively.

It follows immediately that the linear programming relaxations of the strong formulations (14), (2), (6), (15) or (16), (8), (9), (13), (17) always have integral extreme point solutions, and the optimal value of the LP with the makespan objective function (14) or (16) is nothing but the length of the longest chain. Thus it is the labor constraints that make LCSP difficult, and one might naturally conjecture that the less tight the labor constraints, the easier the solution with one of the strong formulations.

2.2 The Block Formulation

As mentioned earlier, LCSP is a simplification of a practical problem arising at BASF. The BASF instances exhibit some special characteristics that prompted the development of formulations tailored to handle these characteritics. The BASF instances have a small set of orders consisting of several jobs with the same labor profiles. Jobs within an order have to be processed one after the other, which leads to precedence constraints defining long paths in the precedence graph. There are only a few precedence constraints between jobs in different orders and it is possible to sort the orders in such a way that this type of precedence constraint only occurs between consecutive orders. The formulation developed to handle the special characteristics of the BASF instances is based on the concept of a *block*.

A block B is a sequence of n_B identical jobs $j_1, \ldots, j_{n_B}$ with $(j_i, j_{i+1}) \in A$, $i = 1, \ldots, n_B - 1$ of the same length and with the same labor profile. We abuse notation by using $e(B) = e(j_1)$, $f(B) = f(j_{n_B})$ for the earliest and latest start times for the jobs in

the block, and $p_B = p_{j_1}$, $\ell_{B,u} = \ell_{j_1,u}$ for the processing times and labor profiles of the jobs in block B.

Throughout this section we assume that the problem is modeled with blocks such that any precedence constraint $(i, j) \in A$ between job i in a block B and job j in a block B', $B \neq B'$ is such that i is the last job in block B and j is the first job in block B'. Thus we obtain a block precedence graph $\overline{D}$ with arc set given by $\overline{A}$. Figure 1 gives an example of the notation we use.

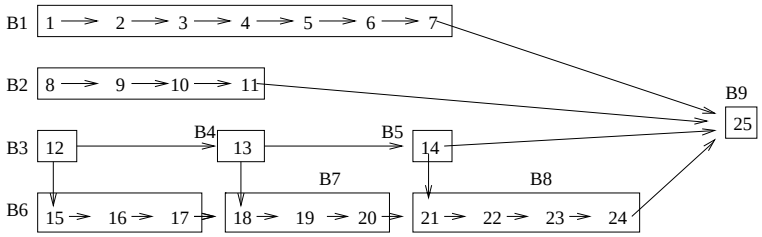

Figure 1: Example of blocks

Define a block variable $X_{B,t} = \sum_{j \in B} x_{j,t}$ so that $X_{B,t} = 1$ if some job of block B starts at time t.

Now the feasible region in the X-space is given by

$$\sum_{t=e(B)}^{f(B)} X_{B,t} = n_B, \qquad \text{for } B \in \mathcal{B} \tag{18}$$

$$X_{B,t} \in \{0, 1\} \qquad \text{for } B \in \mathcal{B}, \;\; t = e(B), \ldots, f(B), \tag{19}$$

using (2) and (6) respectively, where $\mathcal{B}$ is the set of all blocks.

The labor constraint (5) becomes

$$\sum_{B \in \mathcal{B}} \sum_{u=1}^{p_B} \ell_{B,u} X_{B,t-u+1} \leq L \text{ for } t = 1, \ldots, T. \tag{20}$$

Also as only one job in each block can be active in period t, we have the valid inequality

$$\sum_{u=1}^{p_B} X_{B,t-u+1} \leq 1 \text{ for } t = 1, \ldots, T, B \in \mathcal{B}. \tag{21}$$

It is also useful to introduce cumulative block variables. For this, let $Z_{B,t} = \sum_{j \in B} z_{j,t}$ be the number of jobs of block B that have started in or before period t. Note that $Z_{B,t}$ is also the cumulative variable for the block start time variable $X_{B,t}$ so that

$$Z_{B,t} = \sum_{s=e(B)}^{t} X_{B,s}, \quad t = e(B), \ldots, f(B).$$

In the absence of precedence constraints between blocks and labor constraints, (18), (19) and (21) now lead to the formulation (R^Z):

$$\begin{array}{lcl}
Z_{B,f(B)} & = & n_B \text{ for } B \in \mathcal{B} \\
Z_{B,t} - Z_{B,t+1} & \leq & 0 \text{ for } B \in \mathcal{B}, t = e(B), \ldots, f(B) - 1 \\
-Z_{B,e(B)} & \leq & 0 \text{ for } B \in \mathcal{B} \\
Z_{B,t} & \leq & 1 \text{ for } B \in \mathcal{B}, t = e(B), \ldots, e(B) + p_B - 1 \\
Z_{B,t} - Z_{B,t-p_B} & \leq & 1 \text{ for } B \in \mathcal{B}, t = e(B) + p_B, \ldots, f(B).
\end{array}$$

Proposition 2 *The formulation R^Z is integral.*

As for the Proposition 1, the matrix can be easily checked to be totally unimodular.

Using Proposition 1, we can also implicitly describe the convex hull of the set S^Z of feasible solutions to the block formulation in the presence of precedence constraints between blocks.

$$\begin{array}{lcl}
\text{Let } Q = \{(z,Z) : & z_{i,f(i)} = 1 & i = 1, \ldots, n \\
& z_{i,t} - z_{i,t+1} \leq 0 & i = 1, \ldots, n,\ t = e(i), \ldots, f(i) - 1 \\
& -z_{i,e(i)} \leq 0 & i = 1, \ldots, n \\
& -z_{i,t} + z_{j,t+p_i} \leq 0 & \forall (i,j) \in A,\ t = e(i), \ldots, f(i) \\
& \sum_{j \in B} z_{j,t} = Z_{B,t} & \forall B \in \mathcal{B},\ t = e(B), \ldots, f(B)\},
\end{array}$$

and let $Proj_Z(Q) = \{Z : (z,Z) \in Q\}$ denote the projection of the polytope Q onto the space of Z variables.

Proposition 3 $Proj_Z(Q) = conv(S^Z)$.

Finding the projection explicitly may not be simple, though here it is possibly helpful to observe that the dual cone is a network flow with additional constraints. Instead we attempt to derive the inequalities needed to represent precedence constraints between blocks directly. Consider a path of blocks, $B_1, B_2, \ldots, B_r$ with $(B_i, B_{i+1}) \in \bar{A}$ for $i = 1, \ldots, r-1$. We obtain a valid inequality generalizing inequality (21).

Proposition 4 *Given a path of blocks $B_1, B_2, \ldots, B_r$ and $t_1 \geq t_2 \geq \ldots \geq t_r$, the inequality*

$$\sum_{i=1}^{r} (Z_{B_i,t_i} - Z_{B_i,t_i - p_{B_i}}) \leq 1, \tag{22}$$

is valid for S^Z.

Proof: Consider the term $Z_{B_i,t_i} - Z_{B_i,t_i-p_{B_i}}$ for some block $i \in \{1,\ldots,r\}$. We have that $Z_{B_i,t_i} - Z_{B_i,t_i-p_{B_i}} \leq 1$ as each job in block B_i is of length p_{B_i}. Now suppose that $Z_{B_i,t_i} - Z_{B_i,t_i-p_{B_i}} = 1$. Then some job in block B_i starts in the interval $[t_i - p_{B_i}, t_i]$. But then all blocks B_j with $j < i$ must be completed before period t_i, and so $Z_{B_j,t_i-p_{B_j}} = n_{B_j}$. Also all jobs B_j with $j > i$ cannot start until after period t_i and so $Z_{B_j,t_i} = 0$. Now using the fact that the values of the variables $Z_{B,t}$ are nondecreasing, it follows that $Z_{B_j,t_j} - Z_{B_j,t_j-p_{B_j}} = 0$ for all $j \neq i$, and thus the inequality is valid. □

Proposition 5 *Let B and C be two blocks with $(B,C) \in \overline{A}$. The inequality*

$$\sum_{u=1}^{n_C} Z_{B,t+(u-1)p_C} \geq \sum_{u=1}^{n_B} Z_{C,t+up_B+(n_C-1)p_C} \tag{23}$$

is valid for S^Z.

Proof: Let $B = \{b_1,\ldots,b_{n_B}\}$ and $C = \{c_1,\ldots,c_{n_C}\}$. The precedence constraints between jobs imply that for $1 \leq u \leq n_B$ and $1 \leq s \leq n_C$,

$$\begin{array}{rcccccl} z_{c_s,t+up_B+(n_C-1)p_C} & \leq & z_{c_{s-1},t+up_B+(n_C-2)p_C} & \leq & \cdots & \leq & z_{c_1,t+up_B+(n_C-s)p_C} \\ & \leq & z_{b_{n_B},t+(u-1)p_B+(n_C-s)p_C} & \leq & \cdots & \leq & z_{b_{n_B-u+1},t+(n_C-s)p_C} \end{array}$$

Summing over $u = 1,\ldots,n_B$ gives

$$\sum_{u=1}^{n_B} z_{c_s,t+up_B+(n_C-1)p_C} \leq \sum_{u=1}^{n_B} z_{b_{n_B-u+1},t+(n_C-s)p_C} = Z_{B,t+(n_C-s)p_C}.$$

Now summing over $s = 1,\ldots,n_C$ and changing the order of summation gives

$$\begin{aligned} \sum_{u=1}^{n_B}\sum_{s=1}^{n_C} z_{c_s,t+up_B+(n_C-1)p_C} &= \sum_{u=1}^{n_B} Z_{C,t+up_B+(n_C-1)p_C} \\ &\leq \sum_{s=1}^{n_C} Z_{B,t+(n_C-s)p_C} = \sum_{u=1}^{n_C} Z_{B,t+(u-1)p_C}. \end{aligned}$$

□

The results on inequalities for blocks can also be interpreted naturally as valid inequalities in the $x_{j,t}$, or $z_{j,t}$ variables in which all variables in the same block receive the same coefficient.

In the next two sections we attempt to make use of these formulations.

3 LP-based ordering heuristics

Recently there has been much progress on the design and analysis of approximation algorithms for a variety of machine scheduling problems. The design of these approximation algorithms is based on two ideas. The first idea is to use the solution to the linear programming relaxation of some integer programming formulation to suggest an ordering of the jobs and then to construct a feasible schedule consistent with this ordering. The second idea is the notion of an α-point. The α-point of job j, $0 \leq \alpha \leq 1$, is the first point in time at which an α fraction of job j has been completed in the solution to the linear program. The algorithm Schedule-by-Fixed-α orders the jobs by their α-points and schedules them in that order. Goemans [13] has shown that for $1|r_j|\sum w_j C_j$, i.e., the problem of minimizing the sum of the weighted completion times on a single machine subject to release dates, and an appropriate choice of α, this is a $(\sqrt{2}+1)$-approximation algorithm. Goemans also showed that by choosing α randomly according to a uniform distribution and then scheduling in this order, one obtains a randomized 2-approximation algorithm (Schedule-by-Random-α). This algorithm can be derandomized by considering n different values of α, scheduling according to each of them, and then choosing the best (Schedule-by-Best-α). Finally, Schulz and Skutella [24] showed that by randomly choosing n values of α, one value α_j for each job, and ordering the jobs according to these α-points (Schedule-by-Random-α_j), one obtains a 1.693-approximation algorithm. An extensive computational study by Savelsbergh, Uma, and Wein [23] shows that these LP-based ordering heuristics also perform well in practice.

The same two ideas can be used to design heuristics for the LCSP. The α-point for job j is computed in the x-formulations as

$$\text{argmin}_{e(j)\leq s\leq f(j)}\{\sum_{e(j)\leq t\leq s} x_{jt} \geq \alpha\} + p_j$$

and in the z-formulations as $\text{argmin}_{e(j)\leq s\leq f(j)}\{z_{js} \geq \alpha\} + p_j$. Given an ordering of the jobs obtained by one of the α-point schemes, the construction of a feasible schedule consistent with this ordering proceeds in two steps. First, we convert the suggested ordering into another ordering that is consistent with the precedence constraints. This is only necessary when the suggested ordering is derived from an LP solution to one of the weak formulations, because in that case the suggested ordering does not necessarily satisfy the precedence constraints. This new ordering is determined by repeatedly selecting the first available job from the initially suggested ordering, where a job is said to be available if it has either no predecessors in the precedence graph or if all its predecessors in the precedence graph have already been selected. Second, we construct a feasible schedule that respects this ordering, i.e., we assign a start time to each job in such a way that none of the jobs that appear earlier in the ordering have a later start time and such that the precedence and labor constraints are satisfied.

Initial computational experiments indicated that the quality of the schedules obtained in this way was reasonably good. However, higher quality solutions were obtained when these ordering heuristics were combined with simple improvement schemes. We have developed three such improvement schemes.

The first improvement scheme tries to find a better schedule by considering simple modifications to the ordering obtained by an α-point heuristic. It examines every pair of consecutive jobs, checks whether reversing their order violates the precedence relations, and, if not, constructs the schedule associated with this ordering. If an improved schedule is found, this schedule is immediately adopted as the current best schedule and the improvement procedure is repeated.

Note that this type of local search can be thought of as searching in two coupled spaces: the space of the feasible schedules and the space of the job sequences. An advantage of the above scheme is that a small change in a job sequence may result in a large change in the feasible schedule. Such changes might not have been considered if a local search algorithm in the space of feasible schedules was used.

In the second improvement scheme, we take this idea a step further. Let $C_{\max}$ be the makespan of the current schedule. Schedule the jobs in order of nonincreasing completion times as late as possible respecting the precedence and labor constraints and completing at or before $C_{\max}$. A new job sequence is obtained by taking the jobs in order of nondecreasing start times of the just constructed schedule. The rationale for this scheme is the following. Constructing the schedule is a sequential procedure and it may be beneficial to schedule "difficult" jobs as early as possible, when there is still a lot of flexibility. By shifting the jobs forward in time while respecting all the constraints, we hope that the "easy" jobs will naturally move to the end of the schedule, thus leaving the difficult jobs early in the ordering. Note that by applying this idea several times, we may encounter several different job sequences.

The final improvement scheme tries to find an improved schedule by considering changes to the start times of jobs, i.e., the ordering will remain unchanged. Due to the nature of the labor profiles and our schedule construction, the schedules produced are not necessarily semi-active. Consider for example the following instance: $\ell_1 = (4, 1, 1, 2)$, $\ell_2 = (2, 1, 2)$, $\ell_3 = (1, 1, 1)$, $L = 4$, and the ordering (1,2,3). The schedule constructed has $(C_1, C_2, C_3) = (4, 4, 7)$. However, if we force job 2 to start at time 3 instead of 2, we find $(C_1, C_2, C_3) = (4, 5, 5)$. Therefore, it is possible that by starting a job a little later, other jobs can start earlier, possibly resulting in a smaller makespan. This idea has been implemented as follows. For each job j, we construct the schedule that results if we delete job j from the current ordering. Let δ be the change in makespan. If δ is negative, i.e., by deleting job j the makespan decreases, there is a chance that by forcing job j to start later, we may still see a decrease in makespan. Therefore, we construct $|\delta|$ schedules in which we force job j to start at time $s_j + k$ for $k = 1, ..., |\delta|$, where s_j is the start time of job j in the current schedule. This final improvement scheme is used

as a post-processing routine. It is applied to any feasible schedule generated during the solution process.

Each of the four basic formulations, i.e., *weak x-, strong x-, weak z-* and *strong z-formulation*, can be used as the basis for an LP-based ordering heuristic. Note that even though the x- and z-formulations are equivalent, one may have computational advantages over the other, for example in terms of the ease with which the LP relaxation can be solved.

We conducted several computational experiments to see if ordering heuristics provide a viable alternative to other types of heuristics, and to gain some insight in the effect of the choice of formulation on the performance of the ordering heuristics. The chosen formulation affects the ordering heuristic in two ways. First, the strong formulations are significantly larger in size, which may result in linear programming relaxations that are much harder to solve. This may affect the efficiency of the heuristic. Second, the weak formulations produce weaker bounds and therefore linear programming solutions that may provide less useful information to guide the search for high quality schedules. This may affect the effectiveness of the heuristic.

We have used the benchmark instances described in Cavalcante et al. [6] for all our computational experiments. Two of the instances in this set were provided by BASF (*i4o24ja* and *i10o88ja*); the remaining instances were randomly generated so as to resemble the BASF instances. The instance generator takes as input the following parameters: total number of orders (m), minimum and maximum number of jobs in an order (m_j, M_j), minimum and maximum duration of a job in an order (m_d, M_d), and a probability p for two jobs from different orders to have a precedence constraint between them. Based on these input parameters the instance generator constructs an instance with m orders, where each order has between m_j and M_j identical jobs. The jobs in a given order have the same duration, randomly chosen between m_d and M_d. The labor requirement for each period of a job is chosen from the set $\{2, 3, (4), 6, (12), (18)\}$, where the probability that a value between brackets is chosen is 0.1 and the probability that the other values are chosen is 0.3. Precedence relations between jobs in the same order are implicit. Two jobs from consecutive orders have a precedence relation with probability p. A job is successor (predecessor) of at most one job of the previous (next) order. Moreover, for any two consecutive orders, no precedence constraints are redundant. The labor capacities are either 18 or 24. The instances in the benchmark set are denoted by strings of the form $iXoYjZ$, where Y is the number of jobs, X is the number of orders (chains of identical jobs) and Z is an optional character used to differentiate two instances with the same number of orders and jobs.

In the first experiment, we tried to solve the LP relaxation of the instances with 8 orders using CPLEX 5.0 on an IBM RS6000 model 590 with 256 Mb of memory. Table 1 shows the number of rows (#rows), the number of columns (#cols), and the number of nonzeroes (#nz) for each of these instances for the strong x-formulation as well as

the value of the solution to the LP relaxation (lp) and the time it takes to solve the LP relaxation (cpu) in seconds.

Table 1: Size of the strong x-formulation and LP relaxation results.

Name	#rows	#cols	#nz	lp	cpu
i8o63ja	9730	8619	1428143	185.172	14554
i8o63jb	12944	11325	2430918	208.379	41764
i8o63jc	11865	9803	2121477	227.472	12055
i8o65ja	12431	10647	2193483	298.321	10995
i8o65jb[1]	18489	15689	4658174	-	-

[1] Instance could not be solved

Table 2 shows the number of rows, the number of columns, and the number of nonzeroes for each of these instances for the weak x-formulation as well as the value of the solution to the LP relaxation and the time it takes to solve the LP relaxation.

Table 2: Size of the weak x-formulation and LP relaxation results.

Name	#rows	#cols	#nz	lp	cpu
i8o63ja	724	8678	125280	185.170	58
i8o63jb	839	11382	184853	208.379	115
i8o63jc	826	9860	158146	227.412	206
i8o65ja	1035	10709	167472	298.132	248
i8o65jb	992	15751	276370	250.221	175

The differences in solution times are enormous. It is clear that using a standard linear programming solver the strong x-formulation has limited, if any, value in attempting to get schedules for any realistic size instances in a reasonable amount of time. On the other hand, the differences in the values of the LP relaxations are negligible.

In the second experiment, we try to gain insight into the impact of the chosen formulation on the quality of the final schedule. The first part of Tables 3, 4, 5 and 6 shows the value of the solution to the LP relaxation (lp) and for the Schedule-by-Best-α heuristic the length of the schedule produced by the α-heuristic itself (α), the length of the schedule produced by the α-heuristic combined with the three improvement schemes (α^+), and the time it takes to find the latter schedule (cpu). The second part of Tables 3, 4, 5 and 6 shows the same information for the best schedules found if the ordering heuristic is embedded in MINTO, an experimental LP-based branch-and-bound algorithm [18]. We

have allowed a maximum of one hour of computing time. The experiments were done on a PC with a 200 Mhz Pentium II processor with 64Mb of memory.

Note that the choice of formulation may also impact the performance of a branch-and-bound algorithm, because one formulation may allow more natural and effective branching. For example, branching on variables $z_{j,t}$ may lead to a more balanced division of the search space, than branching on the $x_{j,t}$ variables.

Table 3: LP ordering heuristic results for the strong x-formulation.

		root			hour		
Name	lp	α	α^+	cpu	α	α^+	cpu
i4o23ja	54.00	60	58	7.42	59	58	7.42
i4o24ja	58.00	73	68	7.15	70	68	7.15
i4o24ja2	58.00	63	60	3.06	61	60	3.06
i4o24ja3	58.00	59	58	1.61	59	58	1.61
i4o24jb	54.67	75	72	11.15	75	72	11.15
i4o27ja	53.20	71	67	10.67	70	67	10.67
i6o41ja	102.68	158	146	448.81	158	146	448.81
i6o41jb	94.00	121	114	52.14	119	113	544.33
i6o41jc	86.50	143	133	164.26	143	133	164.26
i6o44ja	88.20	128	119	83.61	127	119	83.61
i6o44jb	104.00	159	138	182.89	150	138	182.89

Examining the results in these tables, we make the following observations.

1. When using the strong formulations, the ordering heuristics produce the optimal solution for all the instances with 4 chains of jobs. When using the weak formulations this does not happen all the time.

2. When using the strong formulations the quality of the pure α-heuristics, i.e., without any improvement schemes, tends to be a little better than when using the weak formulations.

3. When the α-heuristic is combined with improvement schemes, the quality of the final schedule does not seem to be affected by the formulation that is used.

4. When the ordering heuristics are embedded in a branch-and-bound algorithm, there is only little improvement in the quality of the schedules produced. This is partly due to the fact that even in an hour of computing time relatively few nodes are evaluated, especially when using the strong formulations.

Table 4: LP ordering heuristic results for the weak x-formulation.

		root			hour		
Name	lp	α	α^+	cpu	α	α^+	cpu
i4o23ja	54.00	66	58	4.32	59	58	4.32
i4o24ja	58.00	75	68	3.10	71	68	3.10
i4o24ja2	58.00	66	60	4.50	61	60	4.50
i4o24ja3	58.00	59	58	0.81	59	58	0.81
i4o24jb	54.67	80	73	7.59	76	72	90.06
i4o27ja	53.05	71	67	4.80	68	67	4.80
i6o41ja	102.67	165	145	43.53	157	145	43.53
i6o41jb	94.00	135	116	25.39	121	113	2222.22
i6o41jc	86.50	139	133	16.70	139	132	1174.26
i6o44ja	88.20	129	120	11.88	127	119	315.88
i6o44jb	104.00	157	140	45.05	149	138	319.94
i8o63ja	185.17	303	273	278.32	303	273	278.32
i8o63jb	208.37	368	330	573.61	368	330	573.61
i8o63jc	227.41	323	310	688.36	323	310	688.36
i8o65ja	298.13	445	413	545.84	445	413	545.84
i8o65jb	250.22	456	407	934.42	456	407	934.42

5. Even when using the weak z-formulation, it is impossible to construct schedules within an hour for most of the instances with 8 orders.

Using only the weak formulations, we have repeated the experiments with the Schedule-by-Random-α_j heuristic. Overall, the quality of the schedules produced is similar.

Next, consider the following scheme to create a *reverse* instance:

1. Reverse the labor profiles of all the jobs.

2. Reverse the direction of all the precedence constraints.

It is not hard to see that the feasible schedules of the original instance and the reverse instance are in one-to-one correspondence [7]. Therefore, we can apply the above described LP-based ordering heuristics to the reverse instance as well. In some cases, this very simple idea leads to improved results. Consider, for example, the reverse instance of the one used to illustrate the third improvement scheme, i.e., $\ell_1 = (2, 1, 1, 4)$, $\ell_2 = (2, 1, 2)$, $\ell_3 = (1, 1, 1)$, $L = 4$, and the reverse ordering (3,2,1). The schedule constructed has $(C_1, C_2, C_3) = (3, 3, 5)$, which is optimal.

Table 5: LP ordering heuristic results for the strong z-formulation.

		root			hour		
Name	lp	α	α^+	cpu	α	α^+	cpu
i4o23ja	54.00	60	58	2.69	59	58	2.69
i4o24ja	58.00	74	68	7.19	71	68	7.19
i4o24ja2	58.00	66	60	4.92	61	60	4.92
i4o24ja3	58.00	59	58	3.01	59	58	3.01
i4o24jb	54.67	78	72	12.86	75	72	12.86
i4o27ja	53.20	73	67	6.71	69	67	6.71
i6o41ja	102.68	159	145	400.90	159	145	400.90
i6o41jb	94.00	123	113	106.45	120	112	2806.83
i6o41jc	86.50	140	132	210.84	140	132	210.84
i6o44ja	88.20	126	119	159.21	126	119	159.21
i6o44jb	104.00	154	138	254.65	151	138	254.65

We have repeated our experiments using the reversed instances and found that even though we did find some improved schedules, the overall results are very similar.

Our computational experiments have shown that the usefulness of LP-based ordering heuristics is restricted to small and medium size instances due to the time required to solve the linear programs. Solving these linear programs is difficult, in part, due to the *makespan* objective. Therefore, for each job j let s_j denote its start time and let ρ_j be the length of the longest path from job j to job n in the precedence graph G, and consider the following objective function

$$\sum_{j=1}^{n} \rho_j s_j + (\max_{j=1,\ldots,n-1}\{s_j + p_j\} + 1) \sum_{j=1}^{n} T\rho_j.$$

An optimal schedule with respect to the above objective function is also optimal with respect to the makespan objective function. This can be verified via a simple proof by contradiction (see [7] for details).

Again, we have repeated our experiments using this alternative objective function and found that even though we were able to evaluate more nodes of the search tree, the overall results are very similar.

Table 7 summarizes the values of the best schedules found by the ordering heuristics in any of the experiments.

The overall conclusion is that LP-based ordering heuristics are capable of producing high quality schedules, but are limited to medium size instances due to the computational effort required to solve the linear programs. Note that Van den Akker, Hurkens,

Table 6: LP ordering heuristic results for the weak z-formulation.

		root			hour		
Name	lp	α	α^+	cpu	α	α^+	cpu
i4o23ja	54.00	60	58	2.64	59	58	2.64
i4o24ja	58.00	74	68	4.61	71	68	4.61
i4o24ja2	58.00	64	60	7.31	61	60	7.31
i4o24ja3	58.00	59	58	2.97	59	58	2.97
i4o24jb	54.67	78	73	13.45	75	72	37.92
i4o27ja	53.05	76	67	7.54	70	67	7.54
i6o41ja	102.67	165	145	313.56	165	145	313.56
i6o41jb	94.00	132	114	102.08	122	112	3574.39
i6o41jc	86.50	137	133	201.73	137	133	201.73
i6o44ja	88.20	125	118	141.34	124	118	141.34
i6o44jb	104.00	157	140	219.06	150	138	3098.61
i8o63ja	185.17	300	276	2636.29	300	276	2636.29

Table 7: Summary of the best results for LP-ordering heuristics.

Name	$C_{\max}$
i6o41ja	142
i6o41jb	112
i6o41jc	130
i6o44ja	117
i6o44jb	137
i8o63ja	270
i8o63jb	323
i8o63jc	303
i8o65ja	409
i8o65jb	398

and Savelsbergh [27] have shown that in certain situations Dantzig-Wolfe decomposition techniques can be applied to alleviate the difficulties associated with the size of time-indexed formulations.

4 Exact MIP Approaches

The results of the previous section indicate that the strong formulations lead to very difficult linear programs. What is more, the LP values of both the weak and strong formulations are typically unaffected by the labor constraints, and the LP value remains equal to the length of the longest path in the precedence digraph. One, but not the only, reason for the ineffectiveness of a direct mixed integer programming approach is the *makespan* objective. MIPs are notoriously difficult to solve with min max objective functions.

Here we attempt to overcome these difficulties and solve at least small instances. First we derive additional valid inequalities making explicit use of the values appearing in the labor profiles and constraints. Then we propose a simple enumerative (branch-and-prune) approach based on the observation that the LPs are simpler when the objective function is modified.

4.1 Strong Cutting Planes

The labor constraints (20) with the block constraints (21) for each block form together a knapsack with GUB (generalized upper bound) structure. Let $(B_i, u_i)_{i=1}^r$, $B_i \in \mathcal{B}$, $1 \leq u_i \leq p_{B_i}$ be a GUB cover with the $\{B_i\}_{i=1}^r$ distinct and $\sum_{i=1}^r \ell_{B_i,u_i} > L$.

Proposition 6 *For each t, the GUB cover inequality*

$$\sum_{i=1}^{r} \sum_{u:\ell_{B_i,u} \geq \ell_{B_i,u_i}} X_{B_i,t-u+1} + \sum_{B \notin \{B_1,\ldots,B_r\}} \sum_{u:\ell_{B,u} \geq \max_i\{\ell_{B_i,u_i}\}} X_{B,t-u+1} \leq r-1 \tag{24}$$

is a valid inequality

Corollary 1 *If $\ell_{B,u} = L$ for some (B,u) and $\ell_{B',u'} > 0$, then (B,u) and (B',u') form a cover. For any path P of blocks not containing block B*

$$\sum_{B' \in P} \sum_{u':\ell_{B',u'}>0} X_{B',t-u'+1} + X_{B,t-u+1} \leq 1 \tag{25}$$

is a valid inequality for all t.

The next inequality (for job or block models) uses much more of the problem structure, but requires very strict conditions.

We consider $k \geq 3$ jobs such that the following hold for some positive integers α, β, ξ, t^*:

1. $\xi = \min_{j=1,\ldots,k-1, u=1,\ldots,p_j}\{\ell_{j,u}\} > 0$;
2. $\ell_{k,u} \geq \beta$ for some $1 \leq u \leq p_k$;
3. For $i = 1, \ldots, k-1$, there exist integers $h_i \geq t^*$ such that $\ell_{i,u} \geq \alpha$ for $u = 1, \ldots, h_i$;
4. $2\alpha > L$, $\alpha + \beta \leq L$, $\alpha + \beta + \xi > L$.

Proposition 7 *For any integer q^* with $t^* \leq q^* \leq \min\{p_j \ : \ j = 1, \ldots, k-1\}$, the inequality*

$$\sum_{i=1}^{k-1} \sum_{u=1}^{\min\{p_i, q^*+h_i\}} x_{j,t-u+1} \leq 1 + y(t) + y(t-t^*) + \ldots + y(t - (\lceil \frac{q^*}{t*} \rceil - 1)t^*) \qquad (26)$$

is valid for any t, where $y(t) = 1 - \sum_{u=1:\ell_{k,u}\geq\beta}^{p_k} x_{k,t-u+1}$.

Proof: For jobs $i = 1, \ldots, k-1$, job i is said to be active in period t if $\sum_{u=1}^{\min\{p_i, q^*+h_i\}} x_{i,t-u+1} = 1$.

We suppose *w.l.o.g.* that jobs $1, 2, \ldots, m$ are active at t for some $m \leq k-1$, and that job $i+1$ starts before i for $i = 1, \ldots, m-1$.

Let $T_0 = \{t, t-t^*, \ldots, t - (\lceil \frac{q^*}{t*} \rceil - 1)t^*\}$.

Thus the left hand side of the inequality takes the value of the number of active jobs m, and the right hand side is $1 + \sum_{\tau \in T_0} y(\tau)$.

We now make a series of simple observations:

Observation 1: As job $i+1$ requires α laborunits for h_{i+1} periods and $2\alpha > L$, $s_i - s_{i+1} \geq h_{i+1}$.

Observation 2: As job m is active, $s_m \geq t - \min\{p_m, q^* + h_m\} + 1$. Thus, $s_m \geq t - q^* - h_m + 1$.

Observation 3: From observations 1 and 2, $s_{m-1} \geq s_m + h_m \geq t - q^* + 1$.

We now associate the interval $T_i = [s_i, \min\{t, s_i + t^* - 1\}]$ with each job for $i = 1, \ldots, m-1$.

Observation 4: All the jobs $m, m-1, \ldots, i+1$ started before job i, are still active in period t, and thus they are active during the interval T_i.

Observation 5: The intervals T_i are disjoint because the last period of T_{i+1} is at most $s_{i+1} + t^* - 1$, and the first period of T_i is s_i. From Observation 1, $s_{i+1} + t^* - 1 \leq s_i - h_{i+1} + t^* - 1 \leq s_i - 1 < s_i$.

Observation 6: During interval T_i for $i = 1, \ldots, m-1$, the labor available for job k is at most $L - \alpha - \xi < \beta$ as job m is active throughout T_i.

Observation 7: The last period $s_{m-1} + t^* - 1$ of T_{m-1} satisfies

$$\begin{aligned} s_{m-1} + t^* - 1 &\geq t - q^* + 1 + t^* - 1 \quad \text{(by Observation 3)} \\ &= t + t^* - q^* \\ &= t - (\tfrac{q^*}{t^*} - 1)t^* \\ &\geq t - (\lceil \tfrac{q^*}{t^*} \rceil - 1)t^*. \end{aligned}$$

Claim: For $i = 1, \ldots, m-1$, there exist distinct integers $f_i \in \{0, \ldots, \lceil \frac{q^*}{t^*} \rceil - 1\}$ such that $y(t - f_i t^*) = 1$.

Proof of the claim: We have shown that the disjoint intervals $T_{m-1}, T_{m-2}, \ldots, T_1$ are of length t^*, except possibly for T_1. We need to show that each of the intervals contains a period of T_0, where these periods are equally spaced at intervals t^* between $t - (\lceil \frac{q^*}{t^*} \rceil - 1)t^*$ and t^*.

We first consider T_{m-1}. If $s_{m-1} \geq t - (\lceil \frac{q^*}{t^*} \rceil - 1)t^*$, then T_{m-1} lies in $[t - (\lceil \frac{q^*}{t^*} \rceil - 1)t^*, t]$. Otherwise $s_{m-1} < t - (\lceil \frac{q^*}{t^*} \rceil - 1)t^*$, but then as $s_{m-1} + t^* - 1 \geq t - (\lceil \frac{q^*}{t^*} \rceil - 1)t^*$ from Observation 7, period $t - (\lceil \frac{q^*}{t^*} \rceil - 1)t^*$ lies in T_{m-1}.

$T_{m-2}, \ldots, T_2$ lie in $[t - (\lceil \frac{q^*}{t^*} \rceil - 1)t^*, t]$.

T_1 either contains t, or is of length t^* and lies in $[t - (\lceil \frac{q^*}{t^*} \rceil - 1)t^*, t]$.

Thus, there exists f_i such that $t - f_i t^* \in T_i$ for $i = 1, \ldots, m-1$.

Now, by Observation 6, less than β units of labor are available in $t - f_i t^*$ and thus $y(t - f_i t^*) = 1$.

Finally, the right-hand side

$$1 + \sum_{r=0}^{(\lceil \frac{q^*}{t^*} \rceil - 1)} y(t - r t^*) \geq 1 + \sum_{i=1}^{m-1} y(t - f_i t^*) \geq m,$$

and the inequality is valid. □

It is a straightforward exercise to extend these inequalities to the block formulation presented in Section 2.

4.2 A Branch-and-Prune Modified Objective Algorithm

This second approach is motivated by the observation that the LP relaxations with the makespan objective are very hard to solve, and also the impression that as the

objective value (best bound) never moves during the branch-and-bound search, branching decisions must be more or less random and incoherent. Objective functions such as $\max \sum_j \sum_t t^r x_{j,t}$ with $r \geq 2$ appear to overcome both these difficulties – the LPs are solved more rapidly, and the best bound decreases steadily during the tree search. To compensate for the changed objective function, some form of complete enumeration is required.

Branch-and-Prune Algorithm
i) Choose an initial (feasible) horizon T.
ii) If an LP solution is feasible and fractional, branch.
iii) If an LP solution is infeasible, prune the node.
iv) If an integer feasible solution is found with makespan $T^* \leq T$, store the solution and reduce the time horizon $T \leftarrow T^* - 1$. (Note that this will make many active nodes infeasible).
v) When all nodes are pruned, the last solution found is optimal.

Obviously this algorithm becomes a heuristic if terminated early. It can also be used to establish a lower bound τ on the makespan by initializing with $T = \tau - 1$.

A naive version of the algorithm can be implemented with a standard MIP system by restarting the branch-and-bound search every time that the horizon is reduced in Step iv).

4.3 Results

Results for the small 4 order instances using the naive branch-and-prune approach and the weak x-formulation are presented in Table 8, in which the first column gives the instance, the second the time horizon T used, the third the makespan of the first feasible solution found (*infeas* means problem shown to be infeasible), the fourth the time taken and the fifth the number of branch-and-bound nodes. The valid inequalities (24) and (25) are used to tighten the formulations. For instance *i4o24ja* we have also added four additional inequalities limiting the number of jobs starting in the first 10, 15, 20 and 25 periods. The time and number of nodes to find a feasible solution or prove infeasibility is considerably less than with the makespan objective $\min s_n$. For example for instance *i4o21j* with $T = 85, 83, 82$, the corresponding times are 13s, 196s and 2074s respectively.

Instance *i4o24ja* has been solved using the block formulation and a selection of the inequalities (26) for values of $q^* = 3, 5, 7$ after the removal of a large number of inactive constraints a priori. With $T = 70$, the initial LP value (longest path) is 58, the value with the cuts (26) added is above 65.3, and a solution of makespan 68 is found and proved optimal after 4.5 hours. If the model is used just to prove that 68 is a lower bound on the optimum makespan, it takes 1.25 hours and 8700 nodes. All results in this section have been obtained running XPRESS-MP version 10.37 on a 166 Mhz Pentium.

Table 8: Branch-and-Prune results for the Modified Objective Enumeration with $r = 2$.

Instance	T	Makespan	secs	nodes
i4o21j	85	84	2	17
	83	82	2	11
	82	Infeas(81)	2	7
i4o23j	60	58	5	22
	58	Infeas(57)	2	0
i4o24ja	75	72	44	190
	71	70	183	917
	62	Infeas(61)	390	1877
i4o24jb	79	78	262	1965
	73	72	>12hrs	
	70	Infeas(69)	5250	7627
	69	Infeas(68)	732	1047
i4o27ja	68	67	2576	4729
	64	Infeas(63)	10262	17111

5 A Tabu Search Algorithm for the labor Constrained Scheduling Problem (LCSP)

In Section 3 we have seen that that the usefulness of LP-based ordering heuristics is limited to small and medium size instances and that higher quality schedules are obtained when LP-based ordering heuristics are combined with simple improvement schemes.

Due to the latter observation, we have decided to explore further the idea of local search. It is well-known that traditional local search algorithms usually do not perform well since they often get stuck in a local optimum. Various techniques have been proposed to overcome this problem and among the most popular are the simulated annealing and the tabu search methods. Since their appearance, these methods have been applied successfully to a wide range of combinatorial optimization problems. We refer to [21] and [22] for a thorough presentation of these and other techniques such as genetic algorithms.

There is no evidence in the literature that one such technique outperforms the others. However, due to our prior experience with tabu search, we have developed an algorithm based on this technique aimed at producing high quality solutions for large instances of LCSP.

Tabu search is a meta-heuristic for solving optimization problems, designed to help local search methods escape the trap of local optimality [10, 11]. The method makes use of a flexible adaptive memory structure and tabu restrictions to drive and constrain the

solution search process. The goal is to explore the solution space in an intelligent way, searching for good, hopefully optimal, solutions.

The basic concepts of tabu search are the following ([30]). When escaping from a local optimum one should allow a move to a neighboring solution even though its value is worse. One obvious difficulty with this idea is that cycling may occur (the search may visit the same solution several times). To avoid cycling, certain solutions or moves can be declared *tabu* or forbidden. Directly comparing the new solution with all previously visited ones would be computationally infeasible due to memory and time requirements. Thus, in practice, to prevent cycling we keep a list of recently visited solutions or recently made movements (the *tabu list*). The number of iterations a movement or a solution remains tabu (*tabu tenure*) is a parameter to be tuned. Since there is no justification to declare a solution to be tabu when it is actually the best solution found to date, the method may also include some *aspiration criteria* that are used to overrule the tabu criteria. Two other general concepts used for local search algorithms are *intensification*, which introduces mechanisms that increase the search effort in promising areas of the search space, and *diversification*, which, in contrast to intensification, incorporates mechanisms to facilitate movement to a different region of the search space.

In this section, we present the details of our tabu search algorithm, denoted TSLCSP and we discuss its performance on the complete set of instances described earlier.

5.1 TSLCSP

In TSLCSP, a schedule is represented by a job sequence S. Starting from an initial schedule, TSLCSP proceeds iteratively, choosing at each iteration the best admissible move from a candidate list. The moves in this list are defined by two types of operations on the job sequence (*Insert* and *Swap*) and only moves leading to feasible schedules are chosen. At the end of each iteration, the current schedule is replaced by the new schedule obtained. TSLCSP returns the best schedule found over all iterations.

Next, we describe the main components of TSLCSP in more detail.

Initial Solution
The starting point for TSLCSP can be one of three different schedules:

- S_1: A schedule generated by a greedy heuristic that iteratively builds a schedule according to a well-defined priority rule.
- S_2: A schedule generated by a schedule set based heuristic that builds either an active, or a non-delay, or a hybrid active and non-delay schedule (see [3] for definitions).
- S_3: A schedule generated by an LP-based ordering heuristic, such as the Schedule-by-Best-α heuristic.

Neighborhood Strategy
The moves used in TSLCSP are based on the representation of the schedule as a job sequence S. There are two types of moves:

- *Insert*(i, j): inserts job j immediately in front of job i in the sequence.
- *Swap*(i, j): interchanges the positions of jobs i and j in the sequence.

So, given a sequence S, a neighbor of S is obtained by either an *Insert* or a *Swap* move. The set of all neighbors of S defines the neighborhood of S which we denote by $N(S)$.

For algorithmic purposes, the size of $N(S)$ is too large ($O(n^2)$). As a first step to reduce the size of the neighborhood, we have decided to consider only moves that result in feasible schedules. Though smaller, this neighborhood is still too large to be completely analyzed at each iteration of TSLCSP. Therefore, we have decided to work with four, even smaller, candidate sets of neighbors for a given sequence S:

- $CS_1(S)$: A set of 10 feasible sequences obtained from S by applying an *Insert* move to a randomly chosen pair of jobs (i, j).
- $CS_2(S)$: A set of 10 feasible sequences obtained from S by applying a *Swap* move to a randomly chosen pair of jobs (i, j).
- $CS_3(S)$: The set of feasible sequences obtained from S by applying an *Insert* move to all pairs of jobs (i, j) for a fixed randomly chosen job j.
- $CS_4(S)$: The subset of feasible sequences obtained from S by applying an *Insert* move to all pairs of jobs (i, j) for a fixed randomly chosen job i.

TSLCSP was tested with each one of the four candidate sets of neighbors defined above. At each iteration, the new sequence chosen is the one with least cost which is not tabu (i.e., free of tabu restrictions).

Tabu Restrictions
TSLCSP labels as tabu-active [12] the attributes of moves that were recently executed and that resulted in schedules with a larger objective function. We have not implemented the usual tabu restriction, since we do not declare a move to be tabu-active when it has improved the objective. In principle, by also declaring these movements to be tabu-active, we would avoid cycling. However, in our experiments we have observed that sometimes this prevents us from finding better solutions.

Moves that contain tabu-active attributes are tabu moves. A tabu move cannot be executed unless an aspiration criterion is satisfied.

The following attributes of moves can be labeled tabu-active in TSLCSP:

- A_1: Job j has been moved.
- A_2: Job j has been inserted immediately in front of job i.
- A_3: Jobs i and j have been swapped.

Tabu restrictions are associated with each of these attributes:

- TR_1: Job j cannot be moved for a certain number of iterations.
- TR_2: Job j cannot be inserted immediately in front of job i for a certain number of iterations.
- TR_3: Jobs i and j cannot be swapped for a certain number of iterations.

Tabu Tenure
Tabu tenure specifies the number of iterations that a particular move is forbidden. In TSLCSP, after some preliminary experiments, we have adopted dynamic tabu tenures [12], one for each tabu restriction:

- TT_1 (for TR_1): An integer randomly chosen between $\lfloor 0.5\sqrt{n} \rfloor$ and $\lceil 0.8\sqrt{n} \rceil$.
- TT_2 (for TR_2): An integer randomly chosen between $\lfloor 1.2\sqrt{n} \rfloor$ and $\lceil 1.5\sqrt{n} \rceil$.
- TT_3 (for TR_3): An integer randomly chosen between $\lfloor 0.9\sqrt{n} \rfloor$ and $\lceil 1.1\sqrt{n} \rceil$.

Aspiration Criterion
The aspiration criterion used in TSLCSP is that a tabu restriction can be overridden whenever the corresponding move leads to a schedule better than the best schedule found so far.

Cost Function
We have used both the minimum makespan objective and the alternative objective introduced in Section 3. For both cost functions, given a sequence S, a feasible schedule is generated as follows. We scan the jobs in the order they appear in S, scheduling them at the earliest possible time while satisfying the precedence and labor constraints.

Termination Condition
TSLCSP terminates when one of the following two conditions are satisfied: either the limit on the maximum total number of iterations is reached, or the limit on the maximum number of iterations without improvement to the best known schedule is reached.

Table 9: Configurations of the TSLCSP algorithm.

Configuration	Candidate Solution Set	Tabu Restriction	Tabu Tenure
TSSPLC_1	$CS1$	$TR1$	$TT1$
TSSPLC_2	$CS1$	$TR2$	$TT2$
TSSPLC_3	$CS2$	$TR1$ applied to jobs i and j	$TT1$
TSSPLC_4	$CS2$	$TR3$	$TT3$
TSSPLC_5	$CS3$	$TR1$	$TT1$
TSSPLC_6	$CS3$	$TR2$	$TT2$
TSSPLC_7	$CS4$	$TR1$	$TT1$
TSSPLC_8	$CS4$	$TR2$	$TT2$

5.2 Computational Results

Out of the many combinations of choices for the candidate set of neighbors, the tabu restrictions, and the tabu tenure, eight configurations were chosen for our computational experiments with TSLCSP. These configurations are summarized in Table 9.

Each configuration in Table 9 was tested for the three types of initial schedules and for the two cost functions. The code of the TSLCSP algorithm was written using the C++ language and compiled with the GNU g++ compiler. The computational results presented here were obtained on a SUN SPARCstation 1000, and all the schedules correspond to a scenario with $L = 18$ units of labor available in each period. The instances we have used are available in [8].

Table 10 shows the makespan of the best schedule found and the CPU time to reach it for each instance of the data set. The results on each line correspond to the TS configuration which produces the solution with the smallest makespan.

Due to the randomization we have introduced in our Tabu implementation, we have run each of the 8 configurations 5 times on each of the 25 instances. We have observed that the gap between the worst and the best solution never exceeds 3%. However based on these tests, we cannot conclude that a particular tabu configuration outperforms the others.

Finally, we observe that in 8 of the 25 instances, the reversed instance led to the best schedule (indicated by a "*" in the second column of Table 10) and that for half of the instances the best solution was found while using the modified objective function.

It is clear that Tabu search is not able to give lower bounds on the optimal solution and therefore to prove optimality. Thus, the only reasonable comparison with the LP algorithms is to look at the upper bounds generated by both approaches. We have checked the CPU times required by our Tabu code to reach a solution with makespan

Table 10: Best solutions obtained with TSLCSP.

Instance	Makespan	Seconds
i4o21ja	**82**	< 1
i4o23ja	**58**	< 1
i4o24ja	**68**	8
i4o24jb	**72**	14
i4o27ja	**67**	9
i6o41ja	**141***	80
i6o41jb	**110***	10
i6o41jc	**128**	10
i6o44ja	**117***	12
i6o44jb	**137**	29
i8o63ja	**261**	133
i8o63jb	**316**	152
i8o63jc	**296***	28
i8o65ja	**406***	165
i8o65jb	**384**	182
i10o84ja	**636**	337
i10o84jb	**556***	478
i10o85ja	**791**	637
i10o87ja	**582***	123
i10o88ja	**460**	510
i10o100ja	**1468**	87
i10o102ja	**1166**	291
i10o106ja	**1094**	1277
i12o108ja	**1277**	859
i12o109ja	**1343***	381

Table 11: Tabu × LP CPU times.

Instance	best LP makespan	Time (s)	Time × 1.75
i4o23ja	58	< 1	< 1.75
i4o24ja	68	9	15.75
i4o24jb	72	< 1	< 1.75
i4o27ja	67	< 1	< 1.75
i6o41ja	145	< 1	< 1.75
i6o41jb	112	32	56.00
i6o41jc	133	< 1	< 1.75
i6o44ja	118	5	8.75
i6o44jb	138	98	171.50
i8o63ja	276	< 1	< 1.75

not worse than the best makespan obtained by the LP ordering heuristics for some instances of the benchmark. These results are presented in Table 11. The Tabu code was run on a Pentium 350 Mhz with 128 Mb of RAM which is about 1.75 times (the ratio between the CPU clocks) faster than the machine used to obtain the LP results in Table 7. The last column of Table 11 takes into account the difference between the machine speeds. It can be seen that usually only a few seconds are needed by the Tabu code to find a solution a solution as good as that found by the LP heuristic.

For purposes of comparison, in Table 12 we reproduce the best results obtained by the Constraint Programming algorithm of Heipcke and Colombani ([16]) as reported in [6]. The columns **LB** and **UB** represent respectively the lower and upper bounds produced by the algorithm while column **Cr. Path** refers to the length of the critical path in the DAG corresponding to each instance. The symbol "*" is used to indicate when the makespan is optimal. The results have been obtained on a Ultra Sparc Station with 512 Mb RAM. Apparently the CPU times needed by the algorithm to find these bounds are small, but the authors of the CP algorithm do not give details.

It is clear that one of the advantages of Constraint Programming is its ability to produce lower bounds which are much better than those arising from the IP formulations from the previous sections (see tables 3, 4, 5 and 6). Optimality of the 5 smallest instances has been proved by the CP algorithm.

On the other hand, comparing with the feasible solutions obtained with the LP-based ordering heuristics (columns α^+ in Table 4), we conclude that here the IP approach produces better solutions. Moreover, from Table 10, we see that the Tabu Search solutions are even better. On the other hand, the Tabu algorithm is unable to generate a lower bound on the optimal makespan.

6 Conclusions

The results in Sections 3 - 5 show clearly that proving optimality of solutions for LCSPs of reasonable size is still a highly challenging problem. Tabu search appears to be the method of choice if the goal is just to find a good feasible solution for a large instance.

Using time-indexed integer programming formulations and a mixed integer solver, it is possible to find good solutions for medium-sized instances, and weak but nontrivial lower bounds for small instances. It remains an open question whether problem-specific preprocessing, branching and variable selection rules, and valid inequalities can be devised that significantly improve the results. The other challenge is to solve the linear programs more rapidly. Here one might consider alternative algorithms such as Lagrangian relaxation with a network flow subproblem, or use of a starting basis from the network flow problem obtained by dropping the labor constraints.

The problem instances tackled here are all highly constrained, and therefore apparently difficult. A computational study of other classes of LCSPs is needed to show if these instances are exceptional or not.

References

[1] E. Aarts and J.K. Lenstra (1997). *Local Search in Combinatorial Optimization*. John Wiley and Sons, Chichester.

[2] M. van den Akker (1994). LP-based solution methods for single-machine scheduling problems. Ph.D thesis, Technical University of Eindhoven.

[3] K.R. Baker (1974). *Introduction to Sequencing and Scheduling*. John Wiley and Sons, New York.

[4] G. Andreatta, L. Brunetta, and G. Guastalla (1988), From Ground Holding to Free Flight: A New Exact Approach. Technical Report, Department of Pure and Applied Mathematics, University of Padova, Italy, undated.

[5] C.C.B.M. Cavalcante and C.C. de Souza (1997). A Tabu Search Approach for Scheduling Problems under Labour Constraints. Technical Report IC-97-13, IC-UNICAMP.

[6] C.C.B.M. Cavalcante, S. Heipcke, Y. Colombani and C. C. de Souza (1997). Scheduling under Labour Resource Constraints: benchmarks. *Submitted*.

[7] C.C.M.B. Cavalcante (1998). Escalonamento com restrição de mão-de-obra: heurísticas combinatórias e limitantes inferiores. MSc. Dissertation. Instituto de Computação. Universidade Estadual de Campinas (UNICAMP), SP, Brazil. *(In portuguese)*.

[8] http://www.dcc.unicamp.br/~cris/SPLC.html.

[9] E. Demeulemeester and W. Herroelen (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem, *Management Science 38*, 1803-1818.

[10] F. Glover (1989). Tabu Search - Part I. *ORSA Journal on Computing 1*, 190-206.

[11] F. Glover (1990). Tabu Search - Part II. *ORSA Journal on Computing 2*, 4-32.

[12] F. Glover and M. Laguna (1995). Tabu Search. C.R. Reeves (ed.). *Modern Heuristic Techniques for Combinatorial Problems*, John Wiley and Sons, 70-150.

[13] M.X. Goemans (1997). Improved approximation algorithms for scheduling with release dates. *Proceedings of the* 8^{th} *ACM-SIAM Symposium on Discrete Algorithms*, 591-598.

[14] H. Gröflin and T. Liebling (1981). Connected and Alternating Vectors: Polyhedra and Algorithms. *Mathematical Programming 20*, 233-244.

[15] S. Heipcke (1995). Resource constrained job-shop scheduling with constraint nets - two case studies. Diploma thesis, Mathematisch Geographische Fakultaet, Katholische Universitaet Eichstaett.

[16] S. Heipcke and Y. Colombani (1997). A New Constraint Programming Approach to Large Scale Resource Constrained Scheduling. Workshop on Models and Algorithms for Planning and Scheduling Problems, Cambridge, UK.

[17] S. Heipcke and Y. Colombani (1997). Solving RCPSPs with SchedEns – Work-in-Progress Report, School of Business, University of Buckingham.

[18] G.L. Nemhauser, M.W.P. Savelsbergh, and G.C. Sigismondi (1994). MINTO, a Mixed INTeger Optimizer, *Operations Research Letters 15*, 47-58.

[19] J.H.Patterson (1984). A Comparison of Exact Approaches for Solving the Multiple Constrained Resource Project Scheduling Problem, *Management Science 30*, 854-867.

[20] A.A.B Pritsker, L.J. Watters, and P.M.Wolfe (1969). Multiproject Scheduling with Limited Resources: a Zero-One Programming Approach, *Management Science 16*, 93-99.

[21] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves and G. D. Smith (1996). Modern Heuristic Search Methods. John Wiley and Sons.

[22] C. Reeves (1993). Modern Heuristic Techniques for Combinatorial Problems, John Wiley and Sons.

[23] M.W.P. Savelsbergh, R.N. Uma, and J. Wein (1998). An Experimental Study of LP-Based Approximation Algorithms for Scheduling Problems, *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 453-462.

[24] A. Schulz and M. Skutella (1997). Scheduling LPs bear probabilities: Randomized Approximations for Min-sum Criteria. *Proceedings of the 1997 European Symposium on Algorithms*, pages 416-429.

[25] H.D. Sherali, Y. Lee, and D.B. Boyer (1995). Scheduling Target Illuminators in Naval Battle-Group Anti-Air Warfare. *Naval Research Logistics Quarterly 42*, 737-755.

[26] J. Sousa and L.A. Wolsey (1992). Time-indexed formulations of non-preemptive single machine scheduling problems. *Mathematical Programming 54*, 353-367.

[27] M. van den Akker, C.A.J. Hurkens, and M.W.P. Savelsbergh (1998). Time-Indexed Formulations for Machine Scheduling Problems: Column Generation. *INFORMS J. on Computing*, to appear.

[28] H.M. Wagner (1969). *Principles of Operations Research*, Prentice-Hall, Englewood Cliffs, Exercise 50, page 502.

[29] L.A. Wolsey (1997). MIP Modelling of Changeovers in Production Planning and Scheduling Problems, *European Journal of Operational Research 99*, 154-165.

[30] L.A. Wolsey (1998). *Integer Programming*, John Wiley and Sons, New York.

Table 12: Constraint Programming results.

Instance	**Cr. Path**	**LB**	**UB**
i4o21jA	78	82	82*
i4o23jA	54	58	58*
i4o24jA	58	68	68*
i4o24jB	54	72	72*
i4o27jA	53	67	67*
i6o41jA	90	109	152
i6o41jB	94	102	110
i6o41jC	81	110	134
i6o44jA	75	98	122
i6o44jB	104	124	149
i8o63jA	174	187	281
i8o63jB	196	239	344
i8o63jC	227	271	344
i8o65jA	298	342	445
i8o65jB	230	315	411
i10o84jA	270	394	730
i10o84jB	200	355	616
i10o85jA	513	671	912
i10o87jA	194	377	610
i10o88jA	362	–	473
i10o100jA	352	830	1587
i10o102jA	550	878	1239
i10o106jA	383	578	1166
i12o108jA	520	838	1412
i12o109jA	819	980	1476