TASK SCHEDULING UNDER GANG CONSTRAINTS

Dirk Christian Mattfeld
Technical University of Braunschweig,
Institute of Business Administration,
Spielmannstr. 8,
38106 Braunschweig,
Germany
d.mattfeld@tu-braunschweig.de

Jürgen Branke
University of Karlsruhe,
Institute AIFB,
76128 Karlsruhe,
Germany
branke@aifb.uni-karlsruhe.de

Abstract In this paper, a short-term manpower planning problem is considered where workers are grouped into gangs to support reliable and efficient operations. The goal is to minimise the total number of workers required by determining an appropriate gang structure, assignment of tasks to gangs, and schedule for each gang. We model such a problem as a multi-mode task scheduling problem with time windows and precedence constraints. While the gang structure and assignment of tasks is optimised by a tabu search heuristic, each gang's schedule is generated by solving the corresponding one-machine scheduling problem by an iterated Schrage heuristic. Because the evaluation of a tabu search move is computationally expensive, we propose a number of ways to estimate a move's impact on the solution quality.

Keywords: manpower planning, multi-mode scheduling, gang constraints, precedence constraints, tabu search, local search, one-machine scheduling.

1. INTRODUCTION

We consider a task scheduling problem as it arises: e.g., from the transhipment of finished vehicles (Mattfeld and Kopfer, 2003). The inter-modal split in the logistics chain requires the intermediate storage of vehicles at a stor-

age area of an automobile terminal. The storage or retrieval of a charge of vehicles forms a task. Since a charge can only be retrieved after it has been stored, precedence constraints between pairs of storage and retrieval tasks are introduced. Temporal constraints with respect to the availability of transport facilities are modelled by time windows.

A task can be processed in different modes determined by the number of drivers executing the task. In order to warrant a safe and reliable relocation of vehicles, drivers are grouped into gangs. The gangs do not change over the course of the planning horizon, typically a time span covering a work shift. Therefore all tasks assigned to one gang are processed in the same mode. The number of gangs as well as their sizes are subject to optimisation, as is the sequence of operations within a task. Time windows of tasks and precedence constraints complicate the seamless processing of tasks within a gang. The objective is to minimise the sum of workers over all gangs established.

As long as the same gang processes two tasks coupled by a precedence constraint, gang scheduling can handle the constraint locally. Whenever a precedence relation exists across gang boundaries, it becomes globally visible, and is modelled as a dynamically changing time-window constraint for its associated tasks. If many precedence constraints exist, the seamless utilisation of manpower capacity within the various gangs is massively hindered by dynamically changing time windows. Managing this constraint will be the greatest challenge while searching for a near-optimal solution to the problem.

In this paper we propose a tabu search procedure which moves single tasks between two gangs. The performance of a move requires the re-scheduling of the two gangs involved. The associated sub-problems are modelled as one-machine problems with heads and tails and varying modes of processing. Since such a sub-problem is already NP-hard for a single mode of processing (Carlier, 1982), an efficient base-heuristic is iteratively applied to determine the manpower demand. Because the evaluation of a move's impact on the solution quality is computationally expensive, the selection of a move in the tabu search heuristic is based on estimates of the move's impact on the manpower.

In Section 2 we discuss related problems and develop a mathematical model. In Section 3 we describe the algorithm in detail, namely, the tabu search framework, the neighbourhood definition, the procedure of scheduling a gang, and, finally, the approximation proposed for selecting a move. We perform a computational investigation for a set of problem parameters in Section 4 before we conclude.

2. RELATED WORK AND PROBLEM MODELLING

We first describe some related problems before we develop a model for the problem considered.

The consideration of multiple modes in the resource constrained project scheduling allows a trade-off between a task's processing time and its resource consumption (Brucker *et al.*, 1999). Mode-identity constraints for prescribed subsets of tasks have been introduced in order to allow the assignment of identical personnel to a group of related tasks (Salewski *et al.*, 1997).

In the audit staff scheduling problem, auditors are assigned to engagements each consisting of various subtasks. All subtasks of an engagement have to be performed by the same auditor in prescribed time windows. The duration of subtasks differ depending on the performing auditor (Dodin *et al.*, 1998).

Besides the apparent analogies to resource constrained project scheduling, there is also a similarity to the vehicle routing problem with time windows. There, a fleet of vehicles serves a number of customers, even though not every vehicle has to be used. The problem is to assign customers to vehicles, and to generate a route for each of the vehicles, such that a performance criterion is optimal (Bramel and Simchi-Levi, 1997, Chapter 7).

Another related problem appears to be the assignment of non-preemptive computing tasks to groups of processors of a multi-processor system (Drozdowski, 1996). The size and number of groups of processors performing a set of tasks can vary, and time windows for the execution of tasks as well as precedence relations between computing tasks exist (Błażewicz and Liu, 1996). The term "gang scheduling" has been introduced in the context of multi-processor scheduling (Feitelson, 1996), but also relates to standard terms of port operations.

We model the logistics problem at hand as a multi-mode gang scheduling problem. Let $\mathcal{A}$ be the set of (non-preemptive) tasks involved in a problem. For each task $j \in \mathcal{A}$, a certain volume V_j is to be processed in a time interval specified by its earliest permissible starting time EST_j and its latest permissible finishing time LFT_j. The predecessor task of task j of an existing pairwise precedence relation is denoted by η_j. If no predecessor is defined, $\eta_j = \emptyset$.

Time is modelled by $1, \ldots, T$ discrete time steps, which are treated as periods rather than as points in time. If one task is complete at time t, its immediate successor task cannot start before $t + 1$.

The workers are grouped into a set of G gangs $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_G\}$. Each gang $\mathcal{G}_i$ is assigned a subset of the tasks $\mathcal{A}_i \subseteq \mathcal{A}$ with $\bigcup_{i=1}^{G} \mathcal{A}_i = \mathcal{A}$ and $\bigcap_{i=1}^{G} \mathcal{A}_i = \emptyset$. The number of workers in gang $\mathcal{G}_i$ is denoted by p_i, the number of tasks assigned to gang $\mathcal{G}_i$ is denoted by $h_i = |\mathcal{A}_i|$. At any time step, a gang can only work on a single task.

A solution is described by the number of workers in each gang, an assignment of tasks to gangs (i.e. a partition of $\mathcal{A}$ into $\mathcal{A}_i$), and a sequence of tasks for each gang.

Let the task on position k in gang i's permutation of tasks be denoted by $\pi_{i,k}$ (i.e. task $\pi_{i,k}$ with $k > 1$ is processed after task $\pi_{i,k-1}$). Starting times

of tasks $s_j \in [1,\ldots,T]$ can be derived from such a task sequence by assuming left-shifted scheduling at the earliest possible starting time. Similarly, the completion times $c_j \in [1,\ldots,T]$ of tasks are fully determined by the starting times and the manpower demand.

The model can be stated as follows:

$$\min \quad \sum_{i=1}^{G} p_i \tag{1}$$

$$s_j \geq EST_j, \qquad \forall j \in \mathcal{A} \tag{2}$$

$$c_j \leq LFT_j, \qquad \forall j \in \mathcal{A} \tag{3}$$

$$s_j > c_{\eta_j}, \qquad \forall j \in \mathcal{A} : \eta_j \neq \emptyset \tag{4}$$

$$s_{\pi_{i,j}} > c_{\pi_{i,j-1}}, \qquad \forall i \in \{1\ldots G\}, j \in \{2\ldots h_i\} \tag{5}$$

$$c_j = s_j + \left\lfloor \frac{V_j}{p_i} \right\rfloor, \qquad \forall i \in \{1\ldots G\}, \forall j \in \mathcal{A}_i \tag{6}$$

$$\mathcal{A}_i \subseteq \mathcal{A} \qquad \forall i \in \{1\ldots G\} \tag{7}$$

$$\bigcup_{i=1}^{G} \mathcal{A}_i = \mathcal{A} \tag{8}$$

$$\bigcap_{i=1}^{G} \mathcal{A}_i = \emptyset \tag{9}$$

Equation (1) minimises the sum of workers p_i over all gangs. Time windows of tasks are taken into account by (2) and (3). Precedence relations among tasks are considered by Eq. (4). Equation (5) ensures a feasible order of tasks belonging to the same gang. The completion time of each task is calculated in (6). Finally, Eqs. (7)–(9) make sure that each task is assigned to exactly one gang.

3. THE TABU SEARCH ALGORITHM

As already stated in the previous section, a solution has to specify three things:

1. The number of gangs G and the number of workers p_i for each gang $\mathcal{G}_i$
2. the assignment of tasks to gangs, and
3. the sequence of operations in each gang.

In this paper, we are going to use a tabu search algorithm to assign tasks to gangs. For every such assignment, inside the tabu search heuristic, an appropriate schedule for each gang is derived by applying a simple Schrage scheduler (Carlier, 1982).

The basic idea of tabu search is to iteratively move from a current solution s to another solution $s' \in \mathcal{N}(s)$ in the current solution's neighbourhood. The neighbourhood definition allows "small" changes to a solution, called moves, in order to navigate through the search space. The move to be executed is selected based on costs $C(s, s')$ associated with the move from s to s'. For minimisation problems, the deepest descent, mildest ascent strategy is applied for selecting moves.

In order to avoid cycling in a local optimum, recently performed moves are kept in a list of currently forbidden ("tabu") moves for a number of iterations. This tabu list is maintained and updated each time a move has been carried out (Glover and Laguna, 1993). We use a variable tabu list length.

In the remaining sections, the following four issues will be addressed in more detail:

- What is a suitable neighbourhood definition?
- How to build an initial solution?
- How to perform a move?
- How to estimate the cost of a move?

3.1 Neighbourhood and Tabu List

The purpose of the tabu search framework is the integration and disintegration of gangs by re-assigning tasks. We have chosen the most elementary move possible, namely the re-assignment of a single task from one gang to another, resulting in a neighbourhood size of roughly $H \cdot G$ with H being the number of tasks involved in a problem instance. We refrain from engaging more complex neighbourhood definitions like the exchange of two tasks between two gangs, because determining the costs $C(s, s')$ for all $s' \in \mathcal{N}(s)$ is computationally prohibitive and, as we will see, the remedy of estimating the costs becomes almost intractable for complex neighbourhoods.

As a move attribute we consider a task entering a gang. Consequently, a tabu list entry forbids a task to leave a certain gang for a certain number of iterations. Not every move is permissible. For example, for a certain interval of the planning horizon, whenever there are more tasks to be scheduled than there are time steps available, the move is excluded from $\mathcal{N}(s)$. Also, moves which disintegrate and integrate a gang at the same time are not considered.

3.2 Building an Initial Solution

The construction of a competitive and feasible solution is not trivial, because the necessary number of gangs is not known. We build an initial solution by separating tasks into as many gangs as possible. Tasks without precedence

relations are placed in a gang of their own, whereas pairs of tasks coupled by a precedence relation are processed by the same gang. In this way, precedence relations can be handled within the local scope of individual gangs, and it is guaranteed that the initial solution is feasible.

For each gang, we determine the minimum number of workers required to process the tasks. For each task j, the number of workers required is $r_j = \lceil V_j/((c_j - s_j) + 1)\rceil$.

In the event that only one task is assigned to a gang i, its minimum number of workers is thus $p_i = \lceil V_j/((LFT_j - EST_j) + 1)\rceil$

If two tasks j and k with $\eta_k = j$ share a gang, the number of workers required is at least as high as the maximum required for each task separately, and at least as high as if the two tasks would be treated as one:

$$p_i \geq \max_{l \in \{j,k\}} \lceil V_l/((LFT_l - EST_l) + 1)\rceil \quad (10)$$

$$p_i \geq \lceil (V_j + V_k)/((LFT_k - EST_j) + 1)\rceil \quad (11)$$

We start with the maximum of these two lower bounds and check for feasibility. If the number of workers is not sufficient to ensure feasibility, we iteratively increase the number of workers by 1 until the schedule becomes feasible.

3.3 Performing a Move

A move is performed in three steps: first, we move the task from one gang to the other. Then, these two gangs are re-scheduled, and contingently their mode (number of workers) is adapted. Finally, it is checked whether the re-scheduling of the two directly involved gangs can also lead to improvements in other gangs due to dynamically changed time windows. These aspects shall be discussed in the following.

Scheduling a Single Gang Let us first assume that the number of workers is given. This problem can be seen as a one-machine scheduling problem with heads and tails, where the head of a task denotes the non-available interval from $t = 1$ to EST_j, and the tail denotes the corresponding interval from $t = LFT_j$ up to the planning horizon T. The head of task no. 5 in Figure 1 ranges from time unit 1 to 8, whereas its tail comprises only time unit 18. Consequently, the time window of task no. 5 covers time units 9–17. In the current mode of processing, two time units are covered.

For our purpose, we extend the notion of heads and tails by the consideration of precedence relations of tasks placed in different gangs. Since only one gang is modified at a time, predecessor and successor tasks placed in other gangs may additionally constrain the temporal placement of tasks. The functions $est()$ and $lft()$ restrict the time window of a task to its currently largest

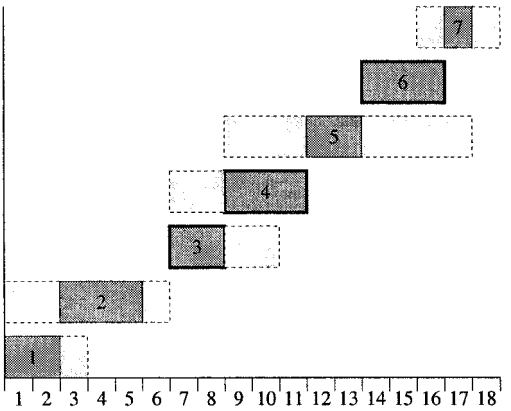

Figure 1. Example of Schrage schedule consisting of seven tasks to be scheduled in 18 time units. Dark grey rectangles represent the time of processing while light grey rectangles depict the time windows given. Critical tasks are indicated by a black border.

permissible extension. $est(j)$ returns EST_j if no predecessor η_j exists and $\max\{EST_j, c_{\eta_j} + 1\}$ otherwise. Similarly, $lft(j)$ returns LFT_j if no successor task κ_j exists and $\min\{LFT_j, s_{\kappa_j} - 1\}$ otherwise.

For the objective of minimising the makespan, this problem has been shown to be NP-hard (Carlier, 1982). Carlier proposes a branch & bound algorithm, which alters a schedule built by the Schrage heuristic and solves the problem optimally. However, because we have to schedule many gangs for each tabu search move, we have to rely on a very fast heuristic. We therefore rely on a single run of the Schrage heuristic which schedules all tasks of $\mathcal{A}_i$ in the planning horizon $1, \ldots, T$ in a given mode p_i.

Basically, the Schrage heuristic $schrage()$ schedules tasks sequentially with respect to the smallest permissible finishing time. In every iteration, one task is placed starting at time unit t. For this purpose, all tasks with $est() \leq t$ enter the selection set $\mathcal{S}$. If $|\mathcal{S}| = 0$, then t is increased to the minimum $est()$ of all tasks not yet scheduled. Otherwise, from $\mathcal{S}$, the task j with the smallest $lft()$ is selected. If it can be placed at time unit t in mode p_i without violating its time window, starting and completion time of j are determined, t is updated and finally j is removed from further consideration.

Figure 1 shows a schedule built by the Schrage heuristic. Initially, tasks no. 1 and 2 can be scheduled at $t = 1$. Task no. 1 is given preference because of its smaller $lft(1) = 3$. In the second iteration only task no. 2 can be placed at $t = 3$. In iteration three, no task is available at $t = 6$ and therefore t is updated to the minimal starting time of the remaining tasks $t = 7$. Task no. 3 dominates no. 4 due to its smaller $lft()$. Then, no. 4 is placed at its latest

permissible time of placement. Finally, the placement of tasks no. 5, 6, and 7 complete the schedule.

The number of workers required is determined in a similar way as for the initial solution described in the previous section. First, let us generalise the calculation of a lower bound to an arbitrary set of tasks $\mathcal{A}_i$. The number of workers required is at least as high as the maximum of the numbers required for any single task, assuming each task can utilise its entire time window (p'). Also, it is at least as high as if the set of tasks is treated as one task (p'').

More formally, the lower bound calculation looks as follows:

function $lower\_bound(\mathcal{A}_i)$
  for all $j \in \mathcal{A}_i$ do $r_j = \lceil V_j/((lft_j - est_j) + 1)\rceil$
  $p' = \max_{j\in\mathcal{A}_i}\{r_j\}$
  $u = \sum_{t=1}^{T} usable(t, \mathcal{A}_i)$
  $p'' = \lceil(\sum_{j\in\mathcal{A}_i} V_j)/u\rceil$
  return $\max\{p', p''\}$
end function

where function $usable()$ returns 1 if time step t can be utilised by at least one task, and 0 otherwise.

As for the initial solution, we first set $p_i = lower\_bound(\mathcal{A}_i)$ and contingently increase p_i by one as long as the Schrage heuristic fails to place all tasks without violating a time window.

procedure $schedule(\mathcal{A}_i)$
  $p_i = lower\_bound(\mathcal{A}_i)$
  while $schrage(\mathcal{A}_i, p_i) =$ `false` do
    $p_i = p_i + 1$
  end while
end procedure

Propagation of Time Windows Scheduling the tasks of a gang may also entail the re-scheduling of other gangs. In the event that tasks of gang i and k involved in the move have precedence constraints with tasks of other gangs, a change of i's or k's schedule may change the time windows of tasks in other gangs, which might lead to different (better or worse) schedules if the Schrage heuristic were applied again.

In particular, we exploit new opportunities due to an enlargement of a time window in the event that the completion time c_j of task j impedes an earlier starting time s_l of the successor task l in another gang. Thus, whenever $c_j+1 = s_l$ holds, and c_j decreases because the gang of task j is re-scheduled, also the gang of task l is noted for re-scheduling. Similarly, whenever the starting time s_l of task l has impeded a later completion at c_j of the predecessor task j

in another gang, that gang is noted for re-scheduling. Once noted, gangs are re-scheduled in a random order.

Since time windows can be recursively extended, we call this procedure *time-window propagation.* The prerequisites for propagating a time window are rarely satisfied, such that the number of re-scheduling activities triggered is limited. If, however, gang i is noted for re-scheduling, there is a reasonable chance to decrease the number of workers p_i.

3.4 Estimating the Cost of a Move

Since the simulation of a move implies at least two, but often many more calls to the Schrage heuristic, it would be computationally very burdensome. Therefore, we estimate a move's effects on the two gangs directly involved and neglect further effects of the propagation of time windows.

To estimate the costs of a move we determine a contingent savings of workers $p_i - \hat{p}_i$ due to the removal of a task j from gang i. Next, we determine the additional effort $\hat{p}_k - p_k$ spent on integrating j into another gang k. We calculate the difference of the two figures, i.e. $\hat{p}_i + \hat{p}_k - p_i - p_k$, and select the move with the highest approximated gain (or the lowest approximated loss) for execution.

Schedule Properties In order to estimate $\hat{p}_i$ and $\hat{p}_k$ for gangs i and k, we discuss some properties of schedules which will help to derive appropriate estimates. Central notions of our arguments are the *block* of tasks and the *criticality* of tasks.

Definition 1 A *block* consists of a sequence of tasks processed without interruption, where the first task starts at its earliest possible starting time, and all other tasks start later than their earliest possible starting time.

Tasks of a block are placed by the Schrage heuristic independent of all other tasks not belonging to this block. Therefore, blocks separate a schedule into several parts, which can be considered independently. Another interesting property concerning blocks is that slack can occur only at the end of blocks or before the first block.

In Figure 1 we identify three blocks. Block 1 consists of tasks no. 1 and 2 and is easily distinguished from block 2 consisting of tasks no. 3, 4, and 5 by the idle-time of time unit 6. Block 3 consisting of tasks no. 6 and 7 can be identified by considering $EST_6 = s_6 = 14$.

Definition 2 A task is said to be *critical* if it is involved in a sequence of tasks (called a critical path), which cannot be shifted back or forth in any way without increasing the number of workers involved.

In a Schrage-schedule, a critical block causes the violation of a time-window constraint if the number of workers is decreased by one. Obviously, all tasks of a critical path belong to the same block, but not every block necessarily contains a critical path. However, if a critical path exists, it starts with the first task of a block. A critical path terminates with the last critical task of its block. Thus, it completes processing at its latest finishing time, although there may exist additional non-critical tasks scheduled later in the same block.

Only one critical path can exist in a block. In the event that a task j scheduled directly before a critical task k causes k's criticality, obviously j itself must be critical. Accordingly, if we classify any task to be critical, all preceding tasks of its block are critical too.

In Figure 1 none of the tasks no. 1 and 2 forming the first block are critical, because the entire block could be shifted to the right by one time unit without delaying other tasks. Tasks 3 and 4 form a critical path within the second block. Although task no. 5 cannot be shifted, it is not critical by definition, because it does not complete at its latest finishing time. Task no. 6 is again critical without the participation of other tasks placed.

As we will see, the notions of blocks and critical tasks make a valuable contribution to the estimation of a move.

Estimating the Manpower Release of a Task Removal Obviously, every schedule has at least one critical block (a block containing a critical path), which impedes a further decrease of the number of workers p_i. For that reason, the only way to obtain a benefit from removing a task from a gang is to break a critical block. If two or more critical blocks exist, and one critical block breaks, at least one other critical block remains unchanged and consequently no benefit can be gained. For instance, in Figure 1 the removal of the block consisting of task no. 6 cannot lead to an improvement because tasks no. 3 and 4 still remain critical. If only one critical block exists, but a non-critical task is removed, again no saving can be expected. In all these cases the estimation procedure returns p_i.

Estimating the Manpower Demand of the Critical Block Removing a critical task from the only critical block existing can lead to a decrease of the number of workers involved. Here we have to distinguish the removal of a task (a) within a critical path, (b) at the beginning of a critical path, and finally, (c) at the end of a critical path.

(a) In case of the removal of a task inside a path we determine the time-span stretching from the starting time s_j of the first task j of the block to the completion time c_l of the last critical task l of the block. We sum the volumes of all tasks but the one to be removed from the critical path, and divide the sum of volumes through the extension of the time-

span. Consider a critical path consisting of tasks 1, 2 and 3. If task 2 is removed, the new number of workers is estimated by $(V_1 + V_3)/((c_3 - s_1) + 1)$.

(b) The removal of the first task j of a critical path alters the starting condition of the path. Therefore the path can start at the maximum of the earliest possible starting time est_l of the second task l of the path, and the completion time $c_m + 1$ of the predecessor task m of task j to be removed (if m does not exists, set $c_m = 0$). For the example of a critical path consisting of tasks 1, 2 and 3, the new number of workers after removing task 1 is estimated by $(V_2 + V_3)/((c_3 - c_{m_1}) + 1)$.

(c) Similarly, the removal of the last task of a critical path alters the terminating condition of the path. Therefore the path can complete at the minimum of the latest possible completion time lft_j of the last but one task j of the path, and the starting time $s_l - 1$ of task l succeeding the path (if l does not exist, set $s_l = T + 1$).

Integrating the Bound Imposed by Non-critical Blocks The approximated manpower demand refers to one critical block only. After the removal of a critical task from this block, other, so far non-critical, blocks may become critical, and for that reason may limit a further decrease of the manpower demand.

Consider a critical block b_c for which the removal of a critical task has been estimated. For blocks in $\mathcal{B} = \{b_1, \ldots, b_{c-1}, b_{c+1}, \ldots, b_n\}$ a lower bound on the number of workers required is calculated by prorating the sum of volumes of its tasks onto the time-span used by the block plus a contingent idle-time following the block. The number of workers applicable is then approximated by the workers $\hat{p}_c$ determined for block c and for the other blocks of $\mathcal{B}$ by $\hat{p}_i = \max\{\hat{p}_c, \max_{k \in \mathcal{B}}\{\hat{p}_k\}\}$.

The procedure accurately identifies the vast majority of non-improving task removals. If the removal of critical tasks in the only critical block existing may lead to a benefit, this benefit is limited by the manpower capacity required for other blocks. In all these cases a conservative estimate $\hat{p}_i$ is returned by the procedure.

Estimating the Manpower Demand of a Task Insertion For estimating the effect on p_k caused by the insertion of a task v into gang k, we first try to fit this task into an existing gang schedule. If v integrates a new gang, we calculate the exact manpower demand p_k of the new gang k as $p_k = \lceil V_j/((lft_j - est_j) + 1)\rceil$. In all other cases, we estimate the additional number of workers $(\hat{p}_k - p_k)$ required in order to produce a feasible schedule.

We start by determining the task w in an existing gang schedule, before which the new task v will be inserted. To identify w, we scan the tasks of the schedule in the order produced by the Schrage heuristic and stop at the first task w with $est_v \leq est_w$ and $lft_v < lft_w$. After having found w, we determine the earliest permissible starting time s_v and the latest permissible completion time c_v with respect to the existing gang structure.

If v has to be appended to the end of the schedule, we easily check whether contingent idle-time $T - c_j$ after the last task j suffices to integrate v.

If v is to be inserted, we are going to verify the available idle-time. Idle-time to the left of w can be available only if w is the first operation of a block. In this case v may start right after the completion time of w's predecessor u, i.e. at time step $c_u + 1$. The utilisation of the idle-time, however, is limited by est_v, thus $s_v = \max\{c_u + 1, est_v\}$.

Idle-time on the right can be available only if w is non-critical. In this case w and its non-critical successor tasks can be shifted to the right in order to obtain additional idle-time. The maximal amount of idle-time available can be determined by placing the tasks right-shifted in the opposite order of the task sequence given in the Schrage schedule. We refer to the resulting starting time of task w as $\overline{s}_w$. Thus, $c_v = \min\{\overline{s}_w - 1, lft_v\}$.

In the event that $\lceil V_v/((c_v - s_v) + 1)\rceil$ is smaller than or equal to the number of workers currently engaged, task v can be integrated in the schedule without engaging additional workers. The procedure returns $\hat{p}_i = p_i$ and terminates.

Whenever the number of workers does not suffice to integrate task v, the additional manpower demand has to be estimated. Since task v for sure becomes critical, the blocks merged by v are considered for prorating v's volume by means of function *lower_bound*(). The approximated increase of the number of workers $\hat{p}_i$ is returned to the calling procedure.

Summary of the Approximation Effort Summarising, the estimation scheme proposed accurately identifies the vast majority of task removals which do not yield savings of workers. In other cases the estimation tends to produce a conservative estimate of savings to be gained. Apart from some special cases, only those insertions are estimated correctly which do not require additional workers. In other cases again a conservative estimate on the additional number of workers required is determined. Whether the approximation quality suffices in order to guide the search successfully will be examined in the following section.

4. COMPUTATIONAL INVESTIGATION

To assess the performance of our tabu search algorithm, we will evaluate it empirically. To that end, we will first propose a benchmark generator, and then

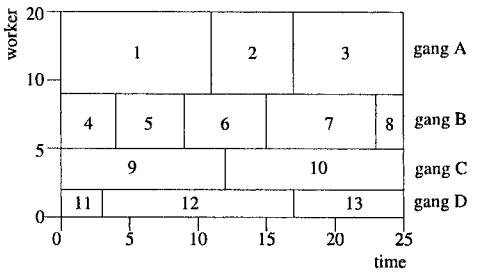

Figure 2. Exemplary optimal solution to a problem instance.

compare our tabu search algorithm with a simple local search approach on a variety of benchmark problems.

4.1 Generating Problem Instances

In this section, we present a way to produce benchmark instances for which the optima are known. The basic idea is to view a schedule as a rectangular plane, where the horizontal extension represents the time dimension and the vertical extension denotes the number of workers involved. Obviously, an entirely filled rectangle means that all workers are occupied for the whole planning horizon, i.e. an optimal schedule.

The example in Figure 2 shows a schedule with 13 tasks organised in 4 gangs. The planning horizon comprises of 25 time units for which 20 workers are utilised. Time windows and precedence relations are not yet specified, but can be easily added to the schedule.

A benchmark instance is generated based on the following input parameters:

- the number of time units in the planning horizon T
- the total number of tasks H
- the number of gangs G.
- the minimal and maximal number of workers in a gang $p_{\min}$ resp. $p_{\max}$
- the percentage of tasks involved in a precedence relation $\gamma \in [0, 1]$
- a parameter $\omega \in [0, 1]$ determining the extension of time windows.

To produce a benchmark instance we proceed as follows:

1. The number of tasks h_i for gang i is determined by prorating the H tasks uniformly upon the G gangs. This can be implemented by first initialising array K of dimension $[0, G]$ and setting $K[0] = 0$ and $K[G] = H$.

Table 1. The mean relative error of the local optima found by the local hill-climber.

	Small problems				Large problems			
γ/ω	1.00	0.75	0.50	0.25	1.00	0.75	0.50	0.25
0.00	2.5	6.3	10.4	18.0	2.1	5.9	10.4	17.3
0.25	2.6	6.5	10.6	16.8	2.1	6.0	10.2	16.2
0.50	2.7	8.1	11.9	17.2	2.3	6.5	10.9	15.9
0.75	3.3	9.9	14.6	18.6	3.1	9.1	13.6	17.1
1.00	5.8	13.3	17.1	20.5	6.0	12.8	16.8	18.7

Then, we assign uniformly distributed random numbers in the range $[1, H-1]$ to $K[1], \ldots, K[G-1]$. Next, we sort K in ascending order, and finally we determine $h_i := K[i] - K[i-1]$.

2. The starting times s_j of task $j \in \mathcal{A}_i$ in gang i are determined by distributing the h_i tasks onto the T time units. We proceed in analogy to the procedure described for Step 1. The completion times c_j of tasks are determined by means of the gang's task chain: $c_{\eta_j} := s_j - 1$.

3. The number of workers p_i of gang i is drawn uniformly distributed from the range $[p_{\min}, p_{\max}]$. Finally we calculate the task volume by multiplying the task's duration with its manpower demand, i.e. $V_j := ((c_j - s_j) + 1) \cdot p_i$.

4. A task can have a precedence relation with every other non-overlapping task of the schedule. For example, in Figure 2 task no. 2 can have a precedence relation with tasks in $\{1, 3, 4, 5, 8, 11, 13\}$. We iteratively select random non-overlapping pairs of tasks not yet involved in a precedence relation and insert a precedence relation until a fraction γ of the tasks are involved in a precedence relation or there is no pair of non-overlapping tasks left.

5. Finally time windows are specified in percent ω of the examined time horizon. In particular, we determine $EST_j := \lceil s_j \cdot \omega \rceil$ and $LFT_j := \lceil c_j + (T - c_j) \cdot \omega \rceil$.

4.2 Empirical Results

To assess the potential of our tabu search procedure, we generate problem instances in two sizes, namely with 10 gangs and 100 tasks (small problems) and with 20 gangs and 200 tasks (large problems). For each size, we additionally vary the extension of time windows ω and the percentage of tasks

Table 2. Mean number of hill-climbing moves performed.

	Small problems				Large problems			
γ/ω	1.00	0.75	0.50	0.25	1.00	0.75	0.50	0.25
0.00	37.4	52.5	59.2	65.9	80.3	104.8	118.5	133.9
0.25	31.7	45.7	51.4	59.3	70.9	91.1	103.1	120.2
0.50	26.0	36.4	42.5	52.7	58.1	78.1	88.1	110.0
0.75	18.8	25.2	33.9	50.2	42.7	54.2	72.9	104.8
1.00	3.8	11.7	28.3	47.7	10.5	27.3	55.8	99.0

involved in precedence relations, γ. Time windows are generated with $\omega \in \{0.25, 0.50, 0.75, 1.00\}$, and the percentage of tasks coupled by a precedence relation are given by $\gamma \in \{0.0, 0.25, 0.50, 0.75, 1.00\}$.

As parameters for the tabu search algorithm, we use a variable tabu list length of $[5, \sqrt{H}]$ where H is the number of tasks involved in the problem, and the stopping criterion is fixed at 10,000 iterations.

For every combination of size, time-window extension and number of precedence relations specified, 10 benchmark instances are generated which are solved three times each, because of the non-deterministic nature of the variable sized tabu list length used. Overall, every figure given in Tables 3 and 4 represents the mean over 30 samples observed.

In order to gauge the competitiveness of the tabu search procedure, additionally a local hill-climber is applied. The algorithm starts from the same initial solution as proposed in Section 3.2 and uses the same neighbourhood definition as the tabu search procedure, refer to Section 3.1. Different from tabu search, the local hill-climber calculates its $C(s, s')$ exactly by simulating all moves in advance. Iteratively, the move yielding the greatest reduction in the number of workers is performed until a local optimum is reached.

For the local hill-climber, the mean relative error (against the optimal solution) over the 10 benchmark instances is presented in Table 1. For unconstrained problems, the relative error is quite small, with 2.5% and 2.1% for small and large instances respectively. However, with an increasing tightness of the constraints imposed, the relative error increases drastically to approximately 20% for $\gamma = 1.00$ and $\omega = 0.25$.

Obviously, the search space becomes more rugged, which goes along with a decreasing performance of the hill-climber. However, as shown in Table 2, the number of hill-climbing moves performed does not directly reflect the ruggedness of the space to be searched. As one may expect, the hill-climbing paths get shorter with an increasing number of precedence constraints imposed (γ). The reasonable relative error of $\approx$6% for $\gamma = 1.0$ and $\omega = 1.0$ is obtained by

Table 3. The mean relative error of the best solutions found by the tabu search procedure.

	Small problems				Large problems			
γ/ω	1.00	0.75	0.50	0.25	1.00	0.75	0.50	0.25
0.00	4.59	8.86	6.87	8.67	4.71	8.76	6.89	8.77
0.25	4.83	8.46	7.42	8.36	4.84	8.20	8.21	9.60
0.50	4.39	7.27	8.15	9.45	5.04	9.47	8.81	9.65
0.75	4.32	7.71	9.93	9.81	4.95	10.32	9.97	10.70
1.00	3.69	9.64	9.19	10.01	4.62	9.16	9.30	10.92

a mere 3.8 moves on average for small problems and 10.5 for large problems respectively.

By narrowing the time windows starting from the entire planning horizon ($\omega = 1.0$) towards 1/4th of the horizon ($\omega = 0.25$), the number of moves performed on a downhill walk increases significantly. Apparently, tight time windows introduce a locality to search such that only tiny improvements per move can be obtained. Although with $\omega = 0.25$ more than 100 moves are performed for large problems, the relative error obtained increases with an increasing tightness of time windows.

Despite performing an increasing number of moves, an increasing relative error is observed. Thus, a further improvement in searching a rugged search space requires the temporary deterioration of the objective function value. Although the tabu search procedure provides this feature, the large number of iterations needed requires a considerably faster estimation of move costs.

Table 3 presents the mean relative error observed for the tabu search procedure. For a maximal time-window extension $\omega = 1.00$ the relative error comprises $\approx$4% regardless of the number of precedence relations specified. This is twice the relative error observed for the hill-climber, and pinpoints at the shortcoming of the estimation procedure. Obviously, the estimation delivers poor approximation of the changes in the number of workers imposed by a move in the case of loose constraints.

Narrowing the time windows increases the relative error of the tabu search procedure only slightly up to $\approx$10% for $\omega = 0.25$. This figure is approximately half of the relative error observed for the hill-climber, which convincingly demonstrates the advantage of tabu search for problem instances with tight constraints.

The tighter the constraints are, the better the costs of a move are approximated by our estimation procedure, because the time span onto which processing times are prorated decreases with an increasing tightness of constraints. Therefore, the estimation procedure is able to guide the search more accu-

Table 4. Number of gangs recorded with the best solution observed.

	Small problems				Large problems			
γ/ω	1.00	0.75	0.50	0.25	1.00	0.75	0.50	0.25
0.00	70.2	30.5	16.2	11.2	138.5	61.3	29.3	20.5
0.25	64.6	29.1	14.1	11.0	125.5	51.3	23.9	19.4
0.50	57.1	20.5	14.6	11.3	115.3	54.4	27.7	19.0
0.75	49.8	18.7	14.9	10.8	100.7	56.2	25.8	18.3
1.00	39.5	28.8	13.3	10.5	82.9	36.1	21.4	18.4

rately. The algorithm proposed seems pleasantly robust against the existence of precedence relations and an increasing problem size.

Table 4 shows the mean number of gangs recorded with the best solution observed. This figure demonstrates how well the gang structure of the optimal solution has been reproduced by the tabu search procedure. For large time windows ($\omega = 1.00$) an enormous number of gangs have been integrated. Since only the integer condition on the number of workers restricts the algorithm from finding the optimal solution, there is no need to reduce the number of gangs. However, merely the existence of precedence relations (which has only a small influence on the solution quality) cuts the number of gangs in half.

For higher constrained problems with $\gamma \geq 0.5$ and $\omega \leq 0.5$ the gang structure of the optimal solution is successfully approximated. Here, for small problem instances $\approx$10 gangs are integrated, whereas the 200 tasks of large problem instances are essentially distributed among $\approx$20 gangs. For highly constrained problems an effective gang structure is a prerequisite for obtaining high quality solutions. Obviously, this structure is identified by the algorithmic approach proposed.

5. CONCLUSION

In this paper, we have addressed the problem of finding a suitable gang structure for a task scheduling problem. For this problem, we have presented a model and we have proposed a way to generate benchmark instances of varying properties. We have developed an efficient tabu search procedure in order to solve such gang scheduling problems.

Particular attention has been paid to the design of the Schrage-scheduler acting as a base-heuristic in the tabu search framework. Although the move of a task from one gang to another modifies just these two gangs directly, the other gangs are indirectly affected by the change of time-window constraints and have to be rescheduled as well.

Since the move neighbourhood is large and the calculation of the cost or benefit is computationally expensive, we have proposed a cost estimation procedure, which approximates the outcome of a move before it is actually performed. Although an estimate must be imperfect in the face of the move's complexity, experience has confirmed the applicability of the estimate developed. For a wide range of benchmark instances a promising solution quality has been achieved.

References

Błażewicz, J. and Liu, Z. (1996) Scheduling multiprocessor tasks with chain constraints. *European Journal of Operational Research*, **94**:231–241.

Bramel, J. and Simchi-Levi, D. (1997) *The Logic of Logistics*. Operations Research Series. Springer, Berlin.

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

Carlier, J. (1982) The one-machine scheduling problem. *European Journal of Operational Research*, **11**:42–47.

Dodin, B., Elimam, A. A., and Rolland, E. (1998) Tabu search in audit scheduling. *European Journal of Operational Research*, **106**:373–392.

Drozdowski, M. (1996) Scheduling multiprocessor tasks—an overview. *European Journal of Operational Research*, **94**:215–230.

Feitelson, D. G. (1996) Packing schemes for gang scheduling. In Feitelson, D. G. and Rudolph, L. (Eds.), *Job Scheduing Strategies for Parallel Processing*, Lecture Notes in Computer Science, Vol. 1162, Springer, Berlin, pp. 89–110.

Glover, F. and Laguna, M. (1993) Tabu search. In Reeves, Colin R. (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell, Oxford, pp. 70–150.

Mattfeld, D. C. and Kopfer, H. (2003) Terminal operations management in vehicle transshipment. *Transportation Research A*, 37.

Salewski, F., Schirmer, A., and Drexl, A. (1997) Project scheduling under resource and mode identity constraints: Model, complexity, methods, and application. *European Journal of Operational Research*, **102**:88–110.

Scheduling in Space