Constraint-Based Methods for Scheduling Discretionary Services [1]

Xiaofang Wang [a,*], Nicola Policella [b], Stephen F. Smith [c] and Angelo Oddi [d]
[a] *School of Business, Renmin University of China, Beijing, China*
E-mail: xiaofang.wang@gmail.com
[b] *European Space Operations Centre, European Space Agency, Darmstadt, Germany*
E-mail: nicola.policella@esa.int
[c] *The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA*
E-mail: sfs@cs.cmu.edu
[d] *ISTC-CNR, Institute for Cognitive Science and Technology, National Research Council of Italy, Rome, Italy*
E-mail: angelo.oddi@istc.cnr.it

Abstract. A project network composed of discretionary tasks typically exists in service professions, such as journalism, clinic, software development or financial analysis, where the quality (or value) of a task increases with the time spent on it. Since a longer task duration consumes more resources (i.e., workers' time), the project manager must strike a balance between quality and time by scheduling tasks and setting their durations while respecting the project deadline, precedence and resource constraints. We formulate this problem, give a polynomial time optimal algorithm for the single capacity case, and prove the NP-completeness of the general multiple capacity case. Then we develop two hybrid solution procedures integrating linear optimization and an AI search procedure — precedence constraint posting — for the general case. Our results verify the effectiveness of these procedures and show there exists a potential synergy between objectives of maintaining temporal flexibility and maximizing quality, which implies exiting techniques in building flexible schedules can be adapted to solve this new class of problems.

Keywords: scheduling, constraint programming, discretionary service

1. Introduction

Most professional work and complex service work, for example, by physicians, software developers, or financial analysts, tend to have discretionary task completion, that is, the task duration is determined by a worker's subjective standards and quality of a task increases with the time spent on it. In resource-constrained environments, where tasks compete for limited resources such as workers' time, the ability to vary task durations provides an additional degree of freedom for resolving resource conflicts. Furthermore, when each task is associated with a duration-dependent quality profile, task durations must be determined to maximize overall quality while respecting project deadlines and resource capacity constraints.

The following examples serve as motivation for our modeling assumptions:

- In a news agency, several different types of activities need to take place before a certain publication date: information gathering, data analysis, writing, editing, confirming and duplicating sources, responding to late-breaking events, etc. All workers (journalists) are professionals and can control durations of these activities based on their urgency/importantce. the overall goal is to maximize the quality of the final issue before the publication date.

[1]This work is a long and integrated version of Wang and Smith [39] and Policella et al. [32].
[*]Corresponding author. E-mail: xiaofang.wang@gmail.com.

- In a software development team, team members cooperate to deliver a software system before a certain deadline. The whole project can be divided into several tasks. Some tasks have high degree of discretion. For example, a team member can decide how much time to spend debugging one piece of code based on his/her judgement of quality. In addition, tasks have interdependencies; debugging an existing piece of code must be finished before coding a new piece.
- In a clinic, a doctor is seeing patients. He can determine the service time for each patient based on his knowledge and subjective judgement. For the same doctor (that is, skill level is fixed), a longer time entails more detailed inquiry and more in-depth diagnosis, which implies higher quality.

Scheduling discretionary services presents an interesting challenge for scheduling research, where in addition to allocating activities to available resources, it is necessary to determine activity durations in a *quality maximizing* manner. In order to capture the quality-time tradeoff in a complex network environment, we model the above processes as a *project*, which must be accomplished before a certain deadline. The project is divided into individual subtasks or activities, each of which requires time and scarce resources for its completion. These activities may have individual *release dates* (earliest start time), and are linked by *precedence constraints* which prescribe the partial order in which activities must be carried out. There is also a *resource constraint* defining the resource capacity, which specifies the maximum resource usage at any given time.

In addition to resource constraints and precedence constraints, a linear *quality profile* is associated with each activity in our problem. This quality profile specifies the quality of the associated activity's output as an increasing function of time. Hence, an activity can be terminated at any time (subject to a required minimum duration, which represents the minimum quality requirement for each activity), with an output quality proportional to its duration. The goal is to establish resource-feasible activity start times and activity durations that maximize overall output quality, given the constraints relating to activity release dates, activity precedence relationships and the project deadline. In the simplest case, overall output quality is defined as the sum of the output quality of each individual activity.

In this paper, we formulate this problem of scheduling discretionary services. We give a polynomial-time optimal algorithm for the unit capacity case and prove the multiple capacity case is NP-complete. Then we develop and empirically evaluate two constraint-based heuristic algorithms incorporating linear optimization and search control heuristics.

Specifically, our heuristic approaches draw on two complementary streams of previous research. The first element of our approach stems from the constraint-based scheduling literature [3,4,8,10,25,26,30,34]. For instance, in Cesta et al. [9] a constraint-based procedure called *Precedence Constraint Posting (PCP)* was shown to be effective in solving difficult instances of RCPSP/Max[1]. The idea underlying *Precedence Constraint Posting (PCP)* [37] is to post supplemental precedence constraints between pairs of activities competing for the same resource until all resource conflicts are eliminated. The *search control heuristics* for the analysis of resource usage, and determination of new constraints are key determinants of the algorithm's performance. In our approach, we adopt this basic framework as a basis for generating a feasible schedule satisfying both resource and precedence constraints.

In order to attain quality maximization, a second element of our approaches is drawn from the classic project management research on the time/cost tradeoff problem (also called project crashing) [23], which addresses the question of whether activity durations should be reduced at some increase in cost. It has been shown [14,22,27] that under the assumption of a linear cost function for each activity, the un-capacitated version of the time/cost tradeoff problem can be modelled as a linear programming problem and efficiently solved to optimality. Noting the similarity between minimizing crashing cost and maximizing quality by adjusting activity durations in a project, we incorporate this linear optimization procedure into our PCP framework to maximize total quality after each new precedence constraint is posted.

In summary, our constraint-based algorithms uses linear optimization to set activity durations, and search control heuristics to determine which sequence constraint should be posted to achieve resource feasibility. A central concept in our heuristics is use of a measure of *flexibility* that combines an activity's reducible duration with its quality profile as a basis for determin-

[1]RCPSP/Max is an extended version of the resource constrained project scheduling problem (RCPSP) formulation considered here that additionally incorporates minimum and maximum time lags between pairs of activities.

ing how to resolve resource conflicts; this concept is found to yield heuristics that exhibit superior performance. To further investigate the impact of retaining flexibility on building a high-quality schedule, we design and evaluate two algorithms which are different in the interaction between these two elements: PCP and linear optimization. In the first algorithm, the linear optimization procedure is *interleaved* with the constraint posting procedure; the activity durations are optimally reset every time a precedence constraint is posted. In the second algorithm, the two goals of maximizing quality and constructing feasible schedules are treated *separately*; a resource feasible schedule is built first, its flexibility is enhanced, and then durations are optimized. Our experimental analysis indicates that there exists a potential *synergy* between objectives of maintaining temporal flexibility and maximizing quality: Retaining flexibility in constructing a feasible schedule may benefit overall quality.

The remainder is organized as follows: We start with an overview of related work in Section 2, followed by the formulation of our problem in Section 3. Then we present our constraint posting algorithms and define a set of constraint-posting heuristics in Section 4. We report the experimental results that indicate the relative performance of our various heuristics in Section 5 and two algorithms in Section 6 respectively. Finally we conclude insights generated from results and contributions of this paper in Section 7. All the proofs are included in the appendix.

2. Related Work

There are several different areas of research that have considered issues relevant to scheduling discretionary services. We review these below.

Time-cost Tradeoff Problems (Project Crashing). Within the operations research community, some work has attempted to bridge the gap between the time-cost tradeoff problem and scheduling under resource constraints, as we do. This work has restricted attention to either the non-preemptive case with discrete cost functions [21,38], or the preemptive scheduling case [35,36]. To our knowledge, there is no work dealing with the problem addressed in this paper, which can be seen as the resource-constrained, non-preemptive time-cost tradeoff problem with linear cost functions.

Anytime Scheduling. Like in our problem, the time of terminating a given activity is decided by the problem solver in research in resource-bounded reasoning [11,20,33,40]. In this context, a tradeoff must be made between output quality and computation time allocated to algorithms, whose results degrade in quality as computation time decreases. But these problems have tended to involve the allocation of a resource with unit capacity. The problems of scheduling so-called “anytime” tasks with both precedence constraints and resource constraints (unit capacity or multiple capacity) have received less attention.

Hybrid Scheduling Techniques. There has been increasing interest in hybrid solutions to scheduling problems, which involve an integration of Constraint Programming (CP) and Linear Programming (LP) techniques [1,5,12,13,17]. Among these, the probe backtrack search procedure in El Sakkout and Wallace [13], which uses LP optimization to continually update a tentative global solution during the search, is quite similar to the approach we adopt in this paper. However, the literature has been principally concerned with the problem of minimum perturbation revision of schedules in response to environmental change (including duration change). We focus instead on the problem of setting durations of discretionary services to maximize quality. In addition, our major interests lie in gaining insight into performance of various search control heuristics that might be integrated into the PCP framework and in investigating the relationship between retaining flexibility and maximizing quality of the schedule.

Temporal Preference Networks. In the temporal preference networks literature [24], the problem of interest is to optimize preference associated decisions such as how long a given activity should last, when it should start, or how it should be ordered with respect to other activities. The preference functions associated with different intervals in these networks are similar to our quality profiles. However, our problem is different from theirs in that we additionally consider resource constraints. The infinite capacity version of our problem can be viewed as one type of temporal reasoning problem with hard constraints (precedence relations) and preferences (each activity prefers to run as long as possible).

Service Operations Management. The quality/speed trade-off in discretionary services has been studied under the formulation of a queueing system in the service operations management literature (for excellent overviews, see 15 and 18): In Hopp et al. [19], an agent decides when to terminate processing a task, where the reward of the task is an increasing function of time. That paper focuses on a *centralized* system in which a real-time task termination policy is determined by the service provider to maximize the average reward rate. Anand et al. [2] studied the similar trade-off in a *decentralized* setting where customers decide whether to use the service based on their evaluation of quality and speed. They assume that the service times are exponentially distributed, but, the value for the customer is a linearly decreasing function of the service rate: shorter services provides less customer value. Anand et al. [2] find the equilibrium joining and pricing strategies. Different from these papers, we focus on the quality/time tradeoff in a more complex project network representing resource and precedence requirements. Due to complexity of this network structure, we ignore stochastic characteristics and focus on solving a deterministic problem.

3. Problem Formulation and Complexity Analysis

Given a project composed of a set of non-preemptive activities $V = \{a_1, \ldots, a_n\}$, a set of precedence constraints, a set of quality profiles and resource with limited capacity, our problem is to decide the start time and the end time of each activity so as to maximize the sum of qualities of all the activities. In the remainder of this paper, we use notations and make assumptions as follows:

- r_i: release date for activity a_i;
- D: a common deadline for all the activities, or the whole project's deadline;
- s_i: decision variable, the start time of activity a_i;
- e_i: decision variable, the end time of activity a_i;
- t: current time;
- q_i: output quality from activity a_i, which is a non-decreasing linear and continuous function of its duration with slope k_i, i. e., $q_i = k_i \cdot (e_i - s_i)$;
- d_i: minimum duration[2] for activity a_i;
- E: the set of edges in the precedence graph, if $(i, j) \in E$, activity a_j should start after activity a_i is completed;
- C: constant resource capacity over the entire horizon. Without loss of generality, each activity is assumed to require one unit of resource;[3]
- Q: objective, the total quality output from the project.

Given a schedule $S = \{(s_i, e_i) : a_i \in V\}$, let

$$A(S,t) := \{a_i \in V \mid s_i \le t < e_i\}(t \ge 0)$$

be the set of activities in progress at time t, also called the *active set* at time t. Let

$$R(S,t) := |A(S,t)|$$

be the amount of resource used at time t. Then for all t, the *resource constraint* is

$$R(S,t) \le C.$$

Given these definitions, the problem of interest can be formulated as follows:

maximize

$$Q = \sum_{i=1}^{n} q_i = \sum_{i=1}^{n} k_i \cdot (e_i - s_i) \tag{1}$$

subject to

$$e_i \le s_j, (i,j) \in E, \tag{2}$$

$$R(S,t) \le C, \tag{3}$$

$$d_i \le e_i - s_i,\ a_i \in V, \tag{4}$$

$$r_i \le s_i,\ a_i \in V, \tag{5}$$

[2]This assumption makes sense in practice because we are required to invest at least some amount of time to each activity in order to guarantee basic quality.

[3]Our model can easily be extended to problems where each activity or task can require more than one unit of resource. In that case, we can divide an activity into several sub-activities, each of which requires one unit of resource and has a quality curve with a smaller slope, and add temporal constraints to synchronize the sub-activities' start and end times.

$$e_i \leq D,\ a_i \in V. \qquad (6)$$

Inequalities (2), (3), (4), (5) and (6) are precedence, resource, minimum duration, release date and due date constraints respectively. The complexity of this problem is established in the following theorem.

Theorem 1. *For the single capacity ($C = 1$) problem, there exists a polynomial algorithm to solve it optimally. However, the multiple capacity ($C > 1$) problem is NP-complete.*

The polynomial algorithm for the single capacity problem is shown in Algorithm 1. The main idea of this algorithm is as follows. Step 2 in the algorithm corresponds to solve the single machine problem given release date and precedence constraints. It finds an optimal schedule with the minimum completion time assuming all activities are at their minimum durations. Step 4 in the algorithm then expands some of the activities in the best way to fill in all the idle periods[4]. In particular, the key to maximize the final schedule's quality is to allocate the idle time to the most valuable activity which is eligible. The complexity of this algorithm is $O(n^2)$. Please refer to the Appendix for the proof of Theorem 1.

In the following section, we briefly summarize the constraint posting procedure underlying our approach and then describe the design of two heuristic algorithms for solving the multiple capacity problem.

```
Input: A set of activities ${a_1, ..., a_n}$ and the constraints.
Output: An optimal schedule.
1. Initialize:
    (a) Durations of activities as their minimum durations;
    (b) Current time $t = \min\{r_i\}, i = 1, ..., n$;
    (c) The un-scheduled activity set $U = \{a_1, \ldots, a_n\}$.
2. while $U \neq \Phi$
    Schedule an activity $a_i \in U$ at $t$ (i. e., $s_i = t$ and $e_i = s_i + d_i$)
    if it satisfies the following two conditions[5]:
    (a) Its release date is reached: $t \geq r_i$;
    (b) All of its predecessors have been finished.
    $U \leftarrow U - \{a_i\}$;
    $t \leftarrow t + d_i$.
3. Set $t = D$.
4. while $t \geq \min\{r_i\}, i = 1, ..., n$
    Search backward from $t$;
    Find the activity $a_j$ which has the maximum quality slope under the condition that there is at least one idle period later than this activity;
    Increase its end time until all the idle periods later than this activity shrink to zero;
    $t \leftarrow s_j$.
5. Return current schedule.
```

Algorithm 1. Algorithm for Single Capacity Problem.

4. Problem Solving via Constraint-based Algorithms

Generally speaking, a Precedence Constraint Posting (PCP) scheduling procedure iteratively transforms a time feasible solution into a resource feasible solution by eliminating all contention peaks. A *contention peak* is a set of activities which simultaneously requires resources in excess of the resource capacity C over a maximal time window $[t_1, t_2]$, where t_1 is the start time of one of the activities in this peak, and t_2 is the end time of one of the activities in this peak. A contention peak is eliminated (or levelled) by posting one or more sequencing constraints between pairs of competing activities in the peak. Each time a new constraint is posted, constraint propagation is performed to update activity start and end times, and to confirm that the solution is still temporally feasible, which means it satisfies all other constraints except the resource constraint. If the schedule is no longer temporally feasible after constraint propagation, the search should either terminate or backtrack to the last choice point and try a different constraint.

In our problem, however, the situation is somewhat different. If we ignore the resource constraint of the problem, we are left with essentially the time-cost tradeoff problem. The goal of this problem is to meet the project's due date, while minimizing total crashing costs. As previously noted, if the cost function for

[4]An idle period is the period of time in which no activities are running; oppositely, the period of time in which an activity is running is called a busy period. So idle period is the time between the end time of a busy period and the start time of next busy period, or between the end time of the last busy period and the project deadline.

[5]If there are multiple activities satisfying the above conditions, they can be scheduled at t in an arbitrary ordering.

each activity is linear, the problem can be optimally solved in polynomial time by linear programming (LP) [14,22,27].

Given this result, we propose and analyze two constraint-based algorithms. Both use linear optimization to set activity durations together with the use of precedence-constraint-posting procedures guided by search control heuristics to determine which sequence constraints should be posted to achieve resource feasibility. More specifically each algorithm exploits a hybrid solution procedure which combines two components: (1) a linear programming (LP) solver for optimally setting the activity durations of a set of temporally related activities ignoring resource constraints, and (2) a precedence constraint posting (PCP) search procedure for resolving resource conflicts and establishing resource feasibility. Within this framework, the dual concerns of resource feasibility and quality optimization can be considered in two different ways.

First, the dual concerns of resource feasibility and quality optimization are considered in a tightly integrated fashion. The LP solver is embedded within the PCP procedure and the impact of various constraint posting decisions on solution quality is used to direct the search process. We call this the "Integrated Hybrid Scheme" in Section 4.1.

Second, motivated by our single capacity algorithm and recent work in the development of procedures for generating temporally flexible schedules [6,8,29], the dual concerns of resource feasibility and quality optimization are considered in a step-wise fashion. Specifically, a precedence constraint posting procedure is first used to generate a resource-feasible, temporally flexible schedule, assuming minimum durations for all project activities. Then, in a second step, we relax these duration constraints and use an LP solver as before to optimally "stretch" project activities while retaining feasibility. We call this the a "Partitioned Hybrid Scheme" in Section 4.2.

4.1. The Integrated Hybrid Scheme

To implement the integrated hybrid scheme, the constraint propagation step of the basic PCP constraint-posting loop was replaced by a call to the linear programming (LP) solver. Thus, new "quality optimal" start times and durations are recomputed each time a new sequencing constraint is posted, and, once all contention peaks are eliminated, the final solution is returned. If the LP solver fails after a constraint has been posted, then the project network has become over-constrained and is no longer temporally consistent. In this case, the search stops and reports failure.[6] The basic schedule generation framework is shown in Figure 1. The algorithm is described in Figure 2.

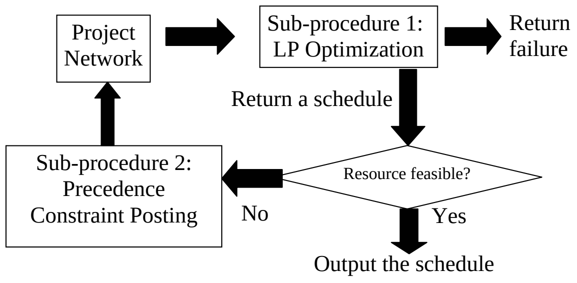


Fig. 1. The Precedence Constraint Posting Framework.

```
Input: A set of activities with constraints.
Output: A solution or failure
1.  loop
2.    apply LP solver to find infinite capacity optimal solution
3.    if there is no temporal feasible solution
4.      then return failure
5.      else begin
6.        detect contention peaks
7.        if there is no peak
8.        then return solution
9.        else constraint posting:
              sequence a pair of activities in some peak
10.     end
11. end-loop
```

Algorithm 2. Quality-Maximizing Constraint Posting Algorithm.

4.1.1. Constraint-posting Heuristics

In this section, we design three classes of heuristics for identifying which activities to sequence in order to resolve conflicts. Because a peak represents a bottleneck in the project network, we always choose activities in some peak to sequence.

Basic Choice Criteria. In designing constraint-posting heuristics, our goal is to post constraints that effectively reduce the resource conflict while minimizing quality loss. Intuitively we prefer to post constraints between activities with the smallest quality **slopes** (k_i),

[6]We concentrate on solving problems without backtracking because our main interest is in evaluating the performance of different search heuristics. Hence, the algorithm is not complete.

because shrinking these activities would seem likely to lead to the smallest quality loss. But slopes only represent the *static* information of activities. A more informed criterion is one that considers the dynamically changing activity durations during the search. To this end, we introduce a second criterion for choosing activities to sequence - choose those activities with the largest **ratios** of "reducible duration" to slope:

$$ratio = \frac{e_i - s_i - d_i}{k_i},$$

where the reducible duration of an activity is the amount greater than its minimum duration. From this choice, we actually bias the choice toward activities with small slope and long reducible duration.

We define two basic heuristics for choosing two activities to sequence based on these criteria, denoted as "Global-slope" and "Global-ratio" respectively in Table 1. In both cases, we assume that once two activities have been selected, they are sequenced according to the rule of *Earliest Start Time First*.

Focusing on the Longest Peak. Observing that an activity's length is closely related to the quality contribution from itself and the other activities running in parallel with it, we can define Longest Peak First (denoted as LPF in the following tables) as an optional preprocessing step that focuses application of either of the basic choice criteria: By using LPF, the pair of activities to sequence is selected from the set of activities in the current longest peak based on ratio or slope.

Heuristics Incorporating Look-ahead. An alternative way to choose a pair of activities is to choose the first activity according to either the criterion of slope or ratio, and to choose the second activity and sequence both activities according to another criterion—**quality loss**. Quality loss is the difference between the two optimal[7] quality values obtained from problems with and without the candidate precedence constraint respectively. The optimal quality will remain unchanged or degrade after posting a constraint. We try to minimize the degradation—quality loss. To compute quality loss exactly, we can run the linear programming solver (LP) twice (before and after posting a given constraint), and this heuristic is denoted as "Ratio-Exact" in Table 1. But this is very costly in computation time. Hence, we design the following two **Quality Loss Estimation** methods.

[7]Here, optimality means the solution for the problem assuming no limits on resource capacity. It is solvable by LP, as we mentioned before.

Loss Only. Suppose we choose the partially overlapped activity pair $< a, b >$. Without loss of generality, we assume they have slopes $k_a \geq k_b$, start times s_a and s_b, end times e_a and e_b, minimum durations d_a and d_b. Based on the relative position of a and b, we can either sequence a before b or b before a, and then shrink a or b or both to eliminate the overlap. If we assume all other activities' start times and end times are fixed in the solution with the newly posted constraint, then the minimum quality loss can be estimated from resolving the resource conflict locally. Given the different potential relative positions of a and b, we use a general form to compute the estimated minimum quality loss. Let A_{loss} be the quality loss due to a's duration change, B_{Loss} be the quality loss due to b's duration change:

$$QualityLoss = A_{loss} + B_{Loss}. \tag{7}$$

If $s_b - s_a \geq e_a - e_b$, then sequence a before b, so

$$A_{loss} = k_a * [max\{e_a + d_b, e_b\} - e_b], \tag{8}$$

$$B_{Loss} = k_b * [min\{e_b - d_b, e_a\} - s_b], \tag{9}$$

If $s_b - s_a < e_a - e_b$, then sequence b before a, so

$$A_{loss} = k_a * [max\{s_b + d_b, s_a\} - s_a], \tag{10}$$

$$B_{Loss} = k_b * [e_b - max\{s_a, s_b + d_b\}]. \tag{11}$$

We illustrate the estimation method with an example in Figure 2. Because $s_b - s_a < e_a - e_b$, we use the formula in equations (10) and (11). As shown in Figure 2, $s_b + d_b > s_a$, so the quality loss can be estimated as follows:

$$A_{loss} = k_a * [s_b + d_b - s_a],$$

$$B_{Loss} = k_b * [e_b - s_b - d_b],$$

$$QualityLoss = A_{loss} + B_{Loss}.$$

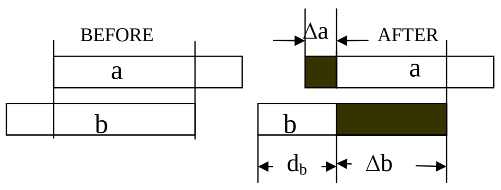


Fig. 2. An Example for Quality Loss Estimation.

From observation, we can see the best quality solution is to sequence b before a, shrink b's duration to minimum duration, and shrink a's duration a little bit to leave enough space for b. This corresponds to the above computation.

Actually, the best quality solutions for all possible relative positions of a and b are generalized in equations (7-11).

If the durations of all other activities remain unchanged after re-optimization of the new problem (with the newly posted constraint), the above quality loss estimation will be the actual quality loss. However, in many cases, if we post a new constraint, other activities will change, too. Thus this method only provides an estimate. This heuristic is denoted as "Ratio-Loss" in Table 1.

Loss and Gain. In the above quality loss function, we ignored some quality gain from a and b's predecessors or successors. If a shrinks by Δa and b shrinks by Δb, a's predecessors or successors can grow longer by up to Δa, similarly, b's predecessors or successors can grow longer by up to Δb. Let the sets of activities which are likely to grow longer be Set_a and Set_b. For example, in Figure 2, Set_a is the set of activity a's predecessors, and Set_b is the set of activity b's successors. The quality gain can be estimated as follows:

$$QualityGain = \sum_{i \in Set_a} k_i * \Delta a + \sum_{i \in Set_b} k_i * \Delta b.$$

If we subtract the quality gain from the above quality loss function, the modified quality loss estimation function is as follows:

$$\begin{aligned} QualityLoss = k_a * \Delta a - \sum_{i \in Set_a} k_i * \Delta a \\ + k_b * \Delta b - \sum_{i \in Set_b} k_i * \Delta b. \end{aligned}$$

This heuristic is denoted as "Ratio-LossGain" in Table 1. We note that the computation of quality gain is overly optimistic because the activities in Set_a and Set_b can be constrained by their other successors or predecessors, and so that they may not be stretchable in all cases. However, quality gain also includes the slope information about this activity's predecessors or successors. It is thus actually an indicator of how "connected" this activity is with other activities.

Table 1
Constraint Posting Heuristics

Heuristics	Basic	LongestPeakFirst	Lookahead
LPF-slope	Slope	Yes	No
Global-slope	Slope	No	No
Slope-Exact	Slope	No	Exact
LPF-ratio	Ratio	Yes	No
Global-ratio	Ratio	No	No
Ratio-Exact	Ratio	No	Exact
Ratio-Loss	Ratio	No	Loss Only
Ratio-LossGain	Ratio	No	Loss and Gain
LPF-Ratio-Loss	Ratio	Yes	Loss
LPF-Slope-Loss	Slope	Yes	Loss

4.2. *The Partitioned Hybrid Scheme: Solve-and-Maximize*

In the partitioned hybrid scheme, the two principal difficulties in solving the resource constrained quality maximization problem are treated separately: first, we establish a *resource-feasible* partial order schedule, assuming minimum durations for all activities; second, these fixed duration constraints are relaxed and *quality optimal durations* are determined.

Before we describe this algorithm in detail, we introduce some concepts about types of scheduling solutions from prior work on building temporally flexible schedules.

Definition 1. A **Fixed Time Schedule** is a schedule in which each activity is allocated a precise start time.

Definition 2. Given a scheduling problem $\mathcal{P}$ and its associated graph $G_P(V_P, E_P)$, where V_P is the set of activities and E_P is the set of precedence constraints connecting these activities, a **Partial Order Schedule**, $\mathcal{POS}$, is a set of solutions that can be represented by a graph $G_{POS}(V_P, E_P \cup E_R)$, where E_R represents the set of newly posted constraints, such that any temporally feasible schedule defined by the graph is also a feasible schedule.

An example taken from Policella et al. [30] is shown in Figure 3 to illustrate these concepts.

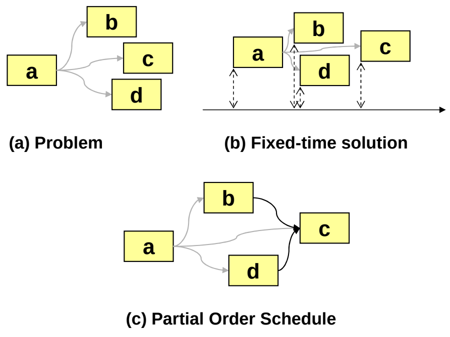


Fig. 3. Types of Scheduling Solutions

Example 1. We have a problem in Figure 3(a), with four activities $\{a, b, c, d\}$, each of which requires one unit of resource. The activity network describes precedence constraints $\{a \prec b, a \prec c, a \prec d\}$, which are represented by grey arrows. Moreover, supposing that the only available resource has maximum capacity of 2, we have a resource conflict among the activities $\{b, c, d\}$. For this problem we consider two alternative solutions: a fixed-time solution as shown in Figure 3(b) and a partial order schedule as shown in Figure 3(c). The fixed time solution consists of a complete activity allocation, that is, each activity is allocated a fixed start time. A partial order solution consists of a further ordering of the subset of activities $\{b, c, d\}$: $\{b \prec c, d \prec c\}$, which are represented by the dark arrows.

4.2.1. Generating a Resource Feasible Partial Order Schedule

In the first phase, to maximize chances of obtaining a resource feasible solution, we simplify the problem. We reduce the degrees of freedom in the problem and assume that all activities will be executed for their minimum duration. Further, to provide greater leverage to the PCP process in leveling conflicts, we restrict attention to the earliest start time solution and concentrate on generating a fixed-times schedule.

Generating a Fixed Time Schedule. The Earliest Start Time Algorithm (ESTA) is a greedy Precedence Constraint Posting approach previously introduced in [7]. It consists of the following three steps: (1) evaluate current schedule; (2) identify possible resource conflicts; and (3) select and solve one conflict by posting a new constraint. These steps are repeated until a solution without any conflict is found. The reason this algorithm is called “Earliest Start Time” is that it analyzes the *earliest start time resource profile* to detect resource conflicts. This earliest start time resource profile represents the resource usage when each activity is allocated at its earliest start time.

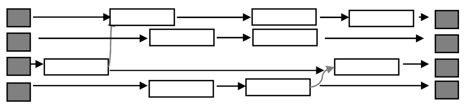


Fig. 4. Chaining-form schedule linking the activities into four chains. Grey arrows represent the precedence constraints defined in the problem, and black arrows represent the posted new constraints.

Once a resource feasible solution is obtained, we must adjust our assumptions to achieve a good quality solution in the second phase. By relaxing the constraint that all activities must execute for their minimum duration only, we can provide the opportunities to expand activity durations. However, since all analysis during schedule generation was focused on the early start time solution, there is no guarantee that we will not reintroduce conflicts as we begin to increase activity durations. We are still hindered by the inflexibility of the fixed-times schedule we have generated. To cope with this problem, the fully instantiated solution is transformed into a $\mathcal{POS}$ [30].

From a Fixed Time Schedule to a Partial Order Schedule. The fixed time schedule is transformed into a $\mathcal{POS}$ by a procedure called chaining. The idea of *chaining* was first proposed in Cesta et al. [7]; it produces a $\mathcal{POS}$ in which a chain of activities is associated with each unit of a resource. This procedure feeds *temporal flexibility* back into the fixed-time solution.

As shown in Figure 4, in a chaining-form representation of the schedule, activities which require the same resource unit are linked via precedence constraints into chains. Given this structure, each constraint becomes more than just a simple precedence. It also represents a producer-consumer relation, allowing each activity to know which other activity will supply the unit of resource it requires for execution. If any slack time (the gap between one activity’s end time and its successor’s start time) exists in the schedule, these chained activities are able to float “back and forth” and/or to execute longer than their minimum duration without creating new resource conflicts. This ability of activities to “float” or “stretch” in response to some perturbation during execution is called *temporal flexibility*.

```
**Chaining**$(P, S)$
**Input**: A problem $P$ and one of its fixed-time schedules $S$
**Output**: A partial order schedule $\mathcal{POS}$
1.  $\mathcal{POS} \leftarrow P$
2.  Sort activities according to their start times in $S$
3.  Initialize all chains empty
4.  for each activity $a_i$
5.      $k \leftarrow SelectChain(a_i)$
6.      $a_k \leftarrow last(k)$
7.      if $\neg\exists(a_k \prec a_i)$
8.          $AddConstraint(\mathcal{POS}, a_k \prec a_i)$
9.      $last(k) \leftarrow a_i$
10. return $POS^{ch}$
```

Fig. 5. Basic Chaining procedure.

The chaining procedure presented in Figure 5, transforms a feasible fixed time schedule into a Partial Order Schedule in chaining form by dispatching activities to specific resource units. The first step sorts all activities according to their start times in the schedule S. Activities are then incrementally allocated to the different chains. The allocation of an activity a_i to a chain k results in adding a precedence constraint (if not already present) between the current last element of the chain, $a_k \leftarrow last(k)$, and a_i, as well as updating the last element of the chain $last(k) \leftarrow a_i$.

The function $SelectChain(a_i)$ is the core of the procedure. Different implementations of $Select\ Chain(a_i)$ may lead to different $\mathcal{POS}$s, and these different $\mathcal{POS}$s can be expected to have different flexibility properties. Note that since the input to a chaining procedure is a resource feasible solution it will always be possible to find the chain that a given activity a_i needs [31].

Generating More Flexible Schedules via Iterative Sampling. As a basic implementation of $SelectChain(a_i)$, the dispatching process is carried out in a specific deterministic manner: The $SelectChain(a_i)$ subprocedure always dispatches the next activity a_i to the first available resource unit (chain). Given an activity a_i, a chain k is *available* if the end time of the last activity allocated on it, a_k, is not greater than the start time of a_i (in the algorithm $a_k \prec a_i$ means that a_k proceeds a_i, that is, the end-time of a_k is not greater than the start-time of a_i). We call this chaining algorithm *Simple Chaining*.

As different $\mathcal{POS}$s can be expected to have different flexibility properties, and thus different quality results, our Partitioned Hybrid Scheme adopts a more sophisticated chaining algorithm, proposed in Policella et al. [28]. This algorithm uses an iterative sampling search procedure to increase the flexibility of the resulting $\mathcal{POS}$. We call this chaining algorithm *Iterative Chaining*. Please see Figure 6 for a basic framework.

```
**Iterative Sampling Procedure**
**Input**: A fixed time schedule $S$
**Output**: A partial order schedule $\mathcal{POS}*$
1. initialize an empty partial order schedule
2. loop
3.     $\mathcal{POS} \leftarrow Chaining_{random}(S)$
4.     if $\mathcal{POS}$ is better than $\mathcal{POS}*$ then
5.         $\mathcal{POS}* \leftarrow \mathcal{POS}$
6.     if termination criterion =*true* then
7.         break
8. return $\mathcal{POS}*$
```

Fig. 6. The Iterative Sampling Procedure.

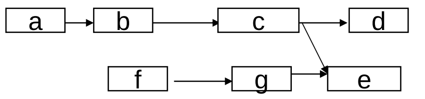
Specifically, the chaining procedure (Figure 5) is executed *numIter* times starting from the same initial fixed-time solution, and randomness is added to the strategy used by $SelectChain(a_i)$ to obtain different $\mathcal{POS}$s across iterations. Each $\mathcal{POS}$ generated is evaluated with respect to some designated measure of flexibility introduced by [7]:

$$fldt = \sum_{i=1}^{n} \sum_{j=1 \wedge j \neq i}^{n} \frac{slack(a_i, a_j)}{H \times n \times (n-1)} \times 100$$

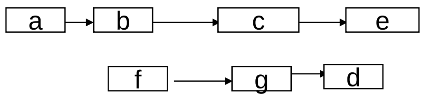
where H is a defined upper bound for the minimum completion time of all activities, n is the number of activities, $slack(a_i, a_j)$ is the width of the allowed interval between the end time of activity a_i and the start time of activity a_j, and 100 is a scaling factor. $fldt$ stands for the *fluidity* of a schedule, i.e., the ability to use flexibility to absorb temporal variation in the execution of activities. So a higher $fldt$ represents a more flexible schedule. The best $\mathcal{POS}$ (according to the above measure of flexibility) found overall is returned at the end of the search.

More precisely, we use a heuristic to produce candidate partial order schedules during the iterative sampling. This heuristic is based on the idea that that if there are fewer constraints/dependencies among different chains, the $\mathcal{POS}$ is more flexible. At the same time, the randomness introduced by this heuristic allows the iterative method to explore more solutions given more iterations.

Specifically, under this heuristic, allocation of an activity a_i proceeds according to the following steps:



(a) A Partial Order Schedule



(b) A More Flexible Partial Order Schedule

Fig. 7. A $\mathcal{POS}$ (top) vs. a more flexible $\mathcal{POS}$ (bottom) generated by the *minimizing the interdependencies* heuristic.

1. The chains satisfying the following condition are collected in the set P_{a_i}:
   - If this chain's last element (activity) is already ordered with activity a_i;
2. If P_{a_i} is not empty, a chain k is randomly selected from P_{a_i}, otherwise a chain k is randomly selected among all available ones.
3. A constraint $a_i \prec a_i$ is posted, where a_k is the last activity on the chain k.

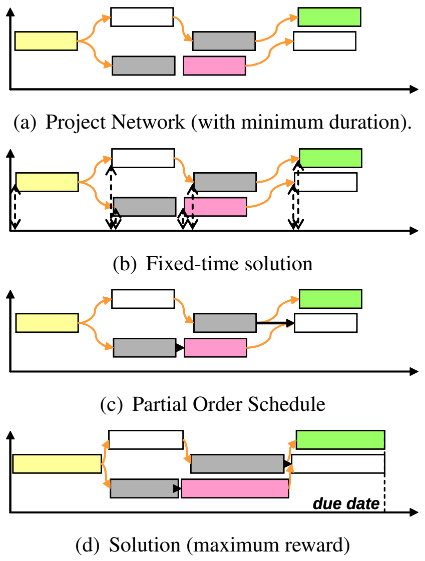
The above heuristic biases the search toward selecting the chain which already contains the predecessor of the current activity, thus the activity can be placed on that chain without introducing a new constraint. Intuitively, a schedule with fewer constraints is more flexible. This heuristic enhances the temporal flexibility of the final chaining-form solution. Please refer to Figure 7 for an example.

4.2.2. *Maximizing Quality*

Once a Partial Order Schedule is obtained, we can expand activity durations by calling Linear Optimization without danger of introducing resource conflict. In fact, the structural properties of a $\mathcal{POS}$ prevent any two activities using same resource unit from overlapping. Of course, duration decisions must still respect the temporal constraints defined by the partial order schedule (i.e., precedence constraints, release date, bounds over the activity durations, common deadline). But this can be accomplished in the same manner as before, by applying the same LP optimization to obtain durations of activities that optimize the overall quality of the $\mathcal{POS}$.

```
Partitioned Hybrid Algorithm
Input: A problem $P$
Output: A solution $s$
1. $P' \leftarrow P \cup_{i=1}^{n} \{e_i - s_i = d_i\}$
2. $s' \leftarrow$find-a-schedule$(P')$
3. $\mathcal{POS} \leftarrow$ Chaining$(P, s')$
4. $s \leftarrow$LPsolver $(\mathcal{POS})$
5. return $s$
```

Fig. 8. The Partitioned Hybrid Algorithm.



(a) Project Network (with minimum duration).

(b) Fixed-time solution

(c) Partial Order Schedule

(d) Solution (maximum reward)

Fig. 9. Steps in the Partitioned Hybrid Algorithm.

We summarize the partitioned hybrid algorithm in Figure 8, and together we use Figure 9 to illustrate the different stages of the Partitioned Hybrid Algorithm.

In Figure 9, the gold (gray) arrows represent the constraints defined in the problem, and black arrows represent those constraints posted during the solution process respectively. First, we compute a fixed-time solution of the problem assuming minimum activity durations (Figure 9(b) and Step 1, Step 2 in Figure 8). This solution is used as the seed schedule for the chaining procedure. The chaining procedure transforms a fixed time schedule into a Partial Order Schedule (Figure 9(c) and Step 3 in Figure 8). Eventually, exploiting the temporal flexibility of a $\mathcal{POS}$, while reserving resource feasibility, the activity durations are stretched by an LP solver to increase the final solution reward as shown in Figure 9(d).

Table 2
Overall Performance Results

Capacity=7	**Quality**	**Runtime**	**Constraints**	**Solved**
Ratio-Exact	96.9	13.8	4.63	392
Ratio-Loss	96.9	0.692	3.97	400
Ratio-LossGain	95.2	0.340	1.29	395
LPF-Ratio	95.2	0.355	1.63	400
Global-Ratio	95.3	0.334	1.62	400
LPF-Slope	97.1	0.378	1.78	396
Global-Slope	96.7	0.468	2.45	386
Slope-Exact	97.3	10.3	4.45	359
LPF-Ratio-Loss	96.0	0.404	1.99	400
LPF-Slope-Loss	95.3	0.416	1.67	393
Capacity=5	**Quality**	**Runtime**	**Constraints**	**Solved**
Ratio-Exact	89.6	31.7	13.0	350
Ratio-Loss	89.5	1.64	11.4	395
Ratio-LossGain	84.0	0.664	3.63	367
LPF-Ratio	84.6	0.938	5.70	400
Global-Ratio	85.0	0.914	5.60	400
LPF-Slope	89.4	0.953	5.93	304
Global-Slope	87.7	1.08	7.05	201
Slope-Exact	89.9	21.5	12.3	217
LPF-Ratio-Loss	87.1	1.09	6.94	399
LPF-Slope-Loss	84.6	0.867	5.12	294
Capacity=3	**Quality**	**Runtime**	**Constraints**	**Solved**
Ratio-Exact	49.5	174.6	65.3	112
Ratio-Loss	49.0	8.44	60.4	228
Ratio-LossGain				41
LPF-Ratio	47.1	3.46	24.6	303
Global-Ratio	47.0	3.49	25.2	291
LPF-Slope				5
Global-Slope				0
Slope-Exact				4
LPF-Ratio-Loss	48.7	4.6	33.1	290
LPF-Slope-Loss				5

5. Evaluation on Performance of Heuristics

In the following two sections we present a set of empirical evaluation results. In the current section, we evaluate the performance of the different heuristics described in Section 4.1.1 under the Integrated Hybrid Scheme. In Section 6, we incorporate the heuristic with the best performance in the Integrated Hybrid Scheme and in the Partitioned Hybrid Scheme respectively to compare performance of these two approaches.

Data Sets. We generated data sets using precedence constraints defined in a resource constraint project scheduling benchmark problem set.[8] For this first evaluation, three data sets were generated with the resource capacities of 3, 5, 7 respectively. Each data set contains 400 problems. Each problem has 32 activities; 32 uniformly distributed random integers in the range $[1, 50]$ were generated to represent the activities' slopes for each problem. We set a global due date $= 30$. Uniformly distributed random integers in the range $[0, 5]$ were generated to represent the release dates for each problem. Uniformly distributed random integers in the range $[1, 3]$ were generated to represent the minimum durations.[9] The algorithms were implemented in Visual C++ 6.0, and LINGO 8.0 was used as the LP solver.[10] The CPU time presented in the following tables were obtained on a Pentium IV-2.40 Ghz processor under Windows XP. In the following tables, we compare the performance of heuristics based on the following measures:

1. *quality(%).* The average percentage quality normalized to the infinite capacity solution; the infinite capacity solution gives us an upper bound for the capacitated problem.
2. *runtime(sec).* The average CPU runtime to reach a solution for one problem.
3. *constraints.* The average number of posted constraints to reach a solution for one problem.
4. *solved(%).* The percentage of problems solved, where a solved problem means the heuristic is able to find a resource feasible solution without backtracking.

The overall performance results are given in Table 2. Because of the fact that some heuristics cannot solve all 400 problems for a given capacity setting, *quality*, *runtime* and *constraints* are calculated based on the commonly solved problems by all heuristics in a given experiment. There are some blanks in Table 2, which is due to the fact that certain heuristics sometimes solve many fewer problems than others, and such a small set of commonly solved problems does not provide the representative information we need for comparison in quality and time.

[8]http://www.bwl.uni-kiel.de/Prod/psplib/data.html. Filename: $j30.sm$.

[9]We choose this range for release dates and minimum durations in order not to make the temporal constraints too tight, and hence guarantee that feasible temporal solutions exist for all the problems.

[10]http://www.lindo.com.

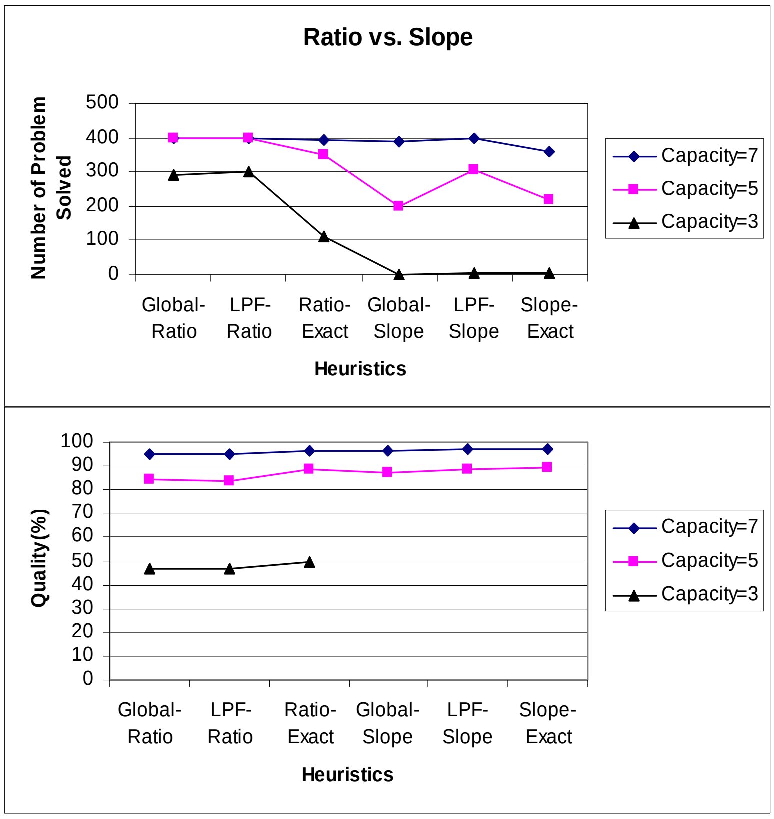


Fig. 10. Comparison between Slope and Ratio.

5.1. Ratio vs. Slope

First, we compare the relative leverage provided by the two basic choice criteria: ratio and slope. Figure 10 depicts the number of solved problems and the average quality of various heuristics. Ratio-based heuristics refer to “Global-Ratio”, “LPF-Ratio” and “Ratio-Exact”. Slope-based heuristics refer to “Global-Slope”, “LPF-Slope” and “Slope-Exact”. We observe the following:

Observation 1. Slope-based heuristics yield slightly better quality performance than ratio-based heuristics on commonly solved problems, however, they solve fewer problems than ratio-based heuristics. Furthermore, on harder (smaller capacity) problems, the number of solved problems using slope-based heuristics degrades significantly while ratio-based heuristics continue to solve most of the problems.

This indicates that there is a tradeoff between solution quality and the percentage of solved problems, which implies a tradeoff between quality and flexibility.

Let us go back to look at the definition of **ratio**. It is the reducible duration divided by quality slope. The larger the reducible duration, the greater the numerator, and the more flexibility for future shrinkage. Let us look at the denominator. The smaller the **slope**, the smaller the denominator, and the lower the quality loss if we shrink this activity. Thus, the ratio represents a balance of the two goals in our search: **maximizing quality and retaining flexibility in the schedule**. The ratio acts to retain flexibility for future shrinking and also achieves good quality.

To understand why this works, it is useful to look at the inflexibility in a typical schedule generated by slope-based heuristics. Consider a resulting schedule using “Global-Slope” when resource capacity is 5. The Gantt chart in Figure 11 illustrates this project schedule. The x-axis represents the time and the y-axis represents the activity IDs. Activities are linked by precedence constraints. The dark bar represents the minimum duration and the grey bar represents the actual duration.

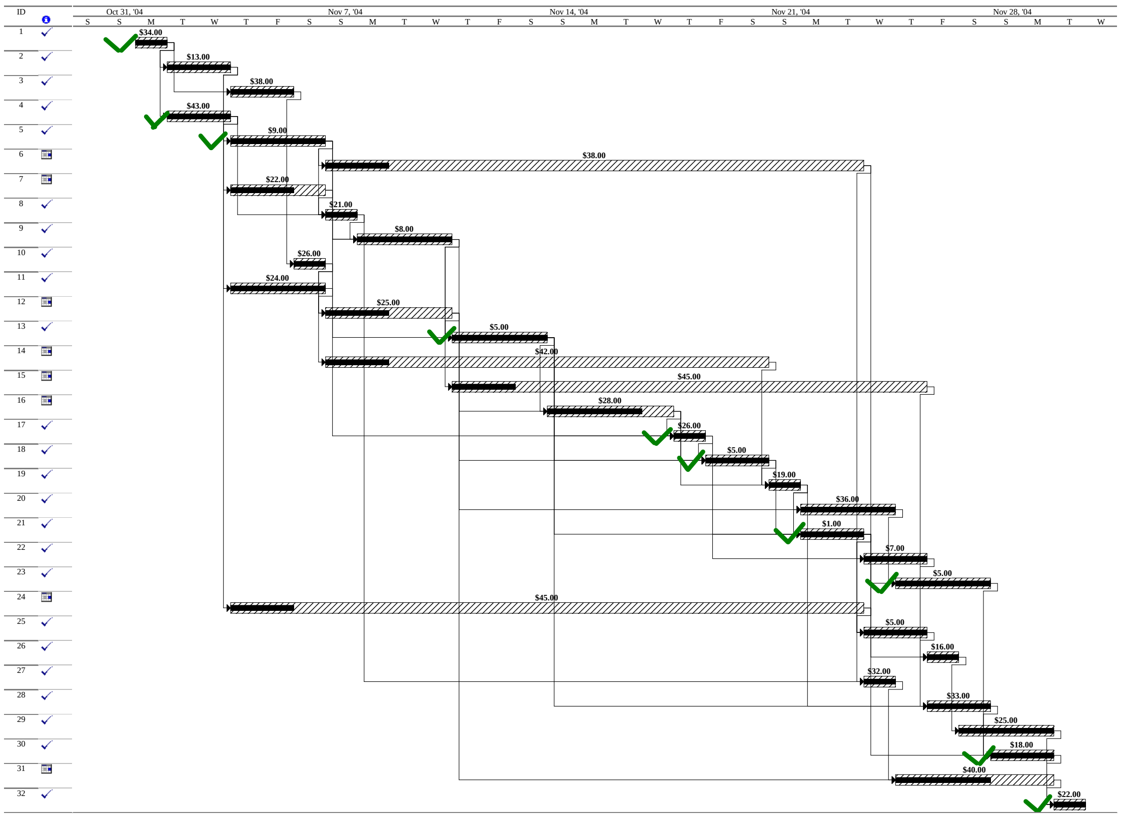


Fig. 11. A Final Schedule Generated by Global Slope.

Table 3

Data for the Activities in the Sequence

Activity ID	1	4	5	13	17	18	21	23	30	32
Duration	1	2	3	3	1	2	2	3	3	1
Min Duration	1	2	3	3	1	2	2	3	3	1
Slope	34	43	9	5	26	5	1	5	18	22

Let us examine more closely a series of sequenced activities depicted as "checked" bars in Figure 11. The IDs of activities in the sequence (ordered by the precedence sequence) are:

$$P = \{1, 4, 5, 13, 17, 18, 21, 23, 30, 32\}$$

Please refer to Table 3 for the data.

Note that all the activities in Table 3 are at their minimum durations and the length (i.e., the sum of activity durations) from the first activity to the last activity is the project duration. We call this sequence of activities a *congested path*. Typically, slope-based heuristics generate many paths like this, which make the schedule inflexible for future potential shrinking: In the constraint posting procedure, slope-based heuristics bias toward sequencing smallest-sloped activities, which are usually placed on these congested paths. After a supplemental constraint is posted, one of the activities sequenced by this constraint must be "squeezed" into a congested path. But since all activities on the path have reached their minimum durations, there is no room for this activity. This leads to a temporally infeasible state and the search terminates. On the other hand, because the small-sloped activities tend to be shrunk until they reach their minimum durations, large-sloped activities can remain relatively longer in the final schedule. So the solution quality from slope-based heuristics is better than that from ratio-based heuristics when in fact the slope-based heuristics succeed.

In contrast to slope-based heuristics, ratio-based heuristics avoid the creation of such congested paths. Because an activity's dynamic duration information is factored into activity selection and long activities are preferred, the search is guided away from a path if it is congested, and instead favors activities on less congested paths. Therefore, the activities at minimum duration are balanced on different paths. Hence, ratio-based heuristics do quite well at avoiding infeasibility.

5.2. *Exact Lookahead or Not?*

We have seen the general advantage of ratio-based heuristics over slope-based heuristics. Now we consider how ratio-based heuristics perform when lookahead with different quality loss computation methods is incorporated. Figure 12 compares those heuristics on solution quality and number of solved problems.

From the results in Figure 12, we can make the following observation.

Observation 2. Local estimation of quality loss—as implemented in"Ratio-loss" heuristics, actually achieves comparable quality to accurate computation ("Ratio-Exact"), while solving more problems and saving significant computational time. "Ratio-Loss" dominates "Ratio-LossGain" in solution quality and number of solved problems.

We explain this observation using a problem instance. Table 4 compares the precedence relations in the resulting schedules from "Ratio-Exact" and "Ratio-Loss" on the same problem instance, when resource capacity is 3. We ignore the activities which have the same set of successors in those two schedules and focus on the differences.

From Table 4, we can see generally that, in the schedule produced by "Ratio-Exact", an activity has more successors than it does in the schedule produced by "Ratio-Loss". Additionally, some activities have very large numbers of successors; for example, activity 11 has 15 successors. The reason is that "Ratio-Exact" tends to sequence two activities with minimum conflict (i.e., minimum overlapping) in order to achieve minimum quality loss. The first activity chosen has the largest ratio, and thus typically has a long duration. It will shrink only a very short amount of time; otherwise, the posted constraint is not the choice achieving minimum quality loss. Thus, this first activity will likely remain the largest ratio activity and be selected again in the next cycle if it is still in some peak. When there is tight capacity, it is very often the case that an activity will remain in its peak after a posting step. So the search process has the following character: the first activity is always the same, and precedence constraints are posted between this activity and many other activities. We call this the "large successor set effect." This search process leads to a schedule which is not flexible, because in this "large successor set," most of

Table 4
Precedence Relations in a Resulting Schedule

Activity ID	Successors in "Exact"	Successors in "Loss"
3	5 7 10 21	4 5 7 21
4	6 7 11 12 13	6 12 14 17 18
	14 17 22 24 27	20 22
5	6 11 12 24 27 29	11 12 14 24 27 29
6	9 12 18 19 20 24	9 16
7	6 12 14 24 27	4 12 14 24 27
8	5 7 13 27	13 27
9	17 19 25	25
10	6 11 12 24 27	4 7 12 24 27
11	9 12 14 15 16 17 18	6 9 16 17 18 22
	19 20 21 22 24 26 27 30	26 30
12	9 18 20	11 18 20
14	20 26 29	11 20 26 29
15	9 12 16 18 19 21 30	6 9 12 16 19 20 30
16	17 18 26 28 30	17 22
17	22 26 29 30 31	22 31
19	21 25	9 21
20	23 25	16 23
21	22 25 26 30	17 22 23
23	28 30	28
27	9 12 14 16 18 20	6 9 11 12 14 16
	21 22 24 28	19 24 28 29

activities are at their minimum durations. For example, among the 15 successors of activity 11 in Table 4, 12 are at their minimum durations. Activity 11 starts at time 8 and its duration is 3, which is also its minimum duration. That means the 12 activities with no flexibility for shrinking must be packed between time $12(= 3 + 8 + 1)$ and time 30 (the project deadline) and satisfy resource capacity constraints. This is a situation very close to temporal infeasibility. And this tendency is why "Ratio-Exact" solves fewer problems than "Ratio-Loss": In seeking optimality it often sacrifices future flexibility.

Furthermore, when both heuristics find feasible schedules, the similarities between the patterns of number of successors, as observed in Table 4, lead to schedules with comparable quality. This is the intuition of why "Ratio-Loss" achieves comparable quality with "Ratio-Exact".

Let us now consider why "Ratio-Loss" is better than "Ratio-LossGain". As we said in the previous section, the computation of quality gain is overly optimistic because activities can be constrained by their other successors or predecessors, such that they may not be stretchable into the space created by posting a sequenc-

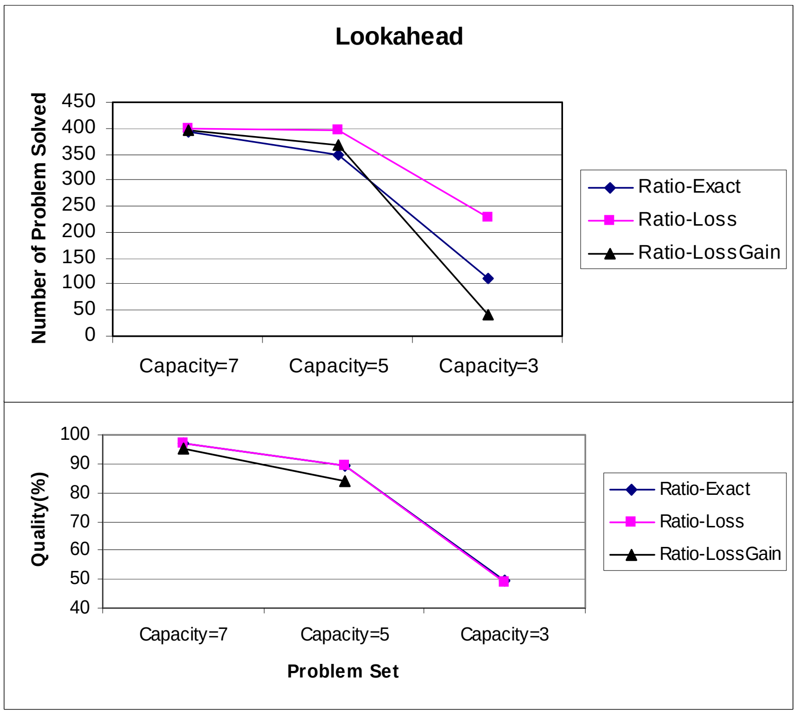


Fig. 12. Different Quality Loss Computation Methods.

ing constraint. From the experimental results, we observe that in our computation of "LossGain", "Gain" is a dominant factor. Without enough consideration on quality loss, therefore, the solution quality becomes worse.

"Ratio-LossGain" tends to sequence activities with lots of successors or predecessors, which increases the severity of the "large successor set effect" as discussed above. Therefore, "Ratio-LossGain" cannot solve many problems. We also observe that "Ratio-LossGain" tends to post fewer constraints overall than either "Ratio-Loss" or "Ratio-Exact". We conjecture that since there is less emphasis on choosing activities that minimize loss, there is a tendency by "Ratio-LossGain" to post constraints that lead to greater reductions in peak length.

5.3. Impact of Longest Peak First(LPF)

Finally, to see the impact of restricting attention to the current longest peak, let us compare "Global-"(without LPF) and "LPF-" heuristics based on the results in Table 2. we observe the following:

Observation 3. "LPF" does not significantly improve quality, but does improve the solvability greatly. This improvement is particularly evident with respect to "Global-slope" and"LPF-slope".

The intuition behind this observation is again the quality/flexibility relationship. Slope-based heuristics statically select the smallest-sloped activities to sequence, leading to a schedule with relatively high quality but one that is also very fragile with respect to future shrinking. "LPF," which selects activities in the longest peak, is a mitigation of this static strategy by taking actual duration information into account, and thus leads to greater flexibility for future shrinkage. In the case of ratio-based heuristics, "LPF" focusing adds additional bias to flexibility. Although ratio-based heuristics are already quite robust with respect to the ability to solve, this additional bias does increase solvability slightly in very tightly constrained problems.

5.4. Summary

All the above observations indicate the important role of temporal flexibility in generating high quality schedules: Heuristics that emphasize quality tend to work against achieving resource feasible solutions due to schedule inflexibility. Therefore, due to a good balance between minimizing quality loss and retaining *flexibility* in each constraint-posting step, "Ratio-Loss" is found to be the most effective heuristic tested. However, in tight capacity problems, its ability of retaining flexibility is limited and thus number of problems solved deteriorates.

Next, we explore further the relationship between quality and flexibility in a framework which is well-known for generating flexible schedules. Specifically, we compare the performance of the integrated hybrid scheme using the winner "Ratio-Loss" as its core constraint-posting heuristic with the partitioned hybrid scheme using iterative chaining.

6. Evaluation of the Two Constraint-based Methods

In this section we evaluate the relative performance of the two approaches described in Section 4: The integrated hybrid scheme using the "Ratio-Loss" heuristic and the partitioned hybrid scheme using *Iterative Chaining*. Our implementation of Iterative Chaining iterates the steps 3 of Figure 8 for 100 times and returns a $\mathcal{POS}$ with the highest flexibility.

As for the same temporal network (i.e.the same set of precedence constraints), a problem can be tightened along two different dimensions: by moving the common deadline earlier or by lowering the resource capacity. For this reason, we consider three different capacity resource values (3, 5, and 7) and three different due dates (25, 30, and 35).

The overall performance results[11] are given in Table 5. There are three points worth mentioning here. First, for the $duedate = 25$ problems, 25 of the 400 problems are known to be infeasible. To provide a fair comparison, *solved* is the percentage of problems solved out of the feasible problems. Second, in cases where one algorithm cannot solve all problems for a given capacity and deadline setting, *quality* and *constraints* are calculated based on the commonly solved problems. Third, there are some blanks in Table 5; these are due to the fact that the integrated approach solves many fewer problems than the partitioned approach, and such a small set of commonly solved problems does not provide the representative information we need for comparison.

[11] In the following comparison CPU times are not presented as the two algorithms were run on two computers with different configurations, but we note that the average solving time on the problems tested for both approaches is on the order of a few seconds.

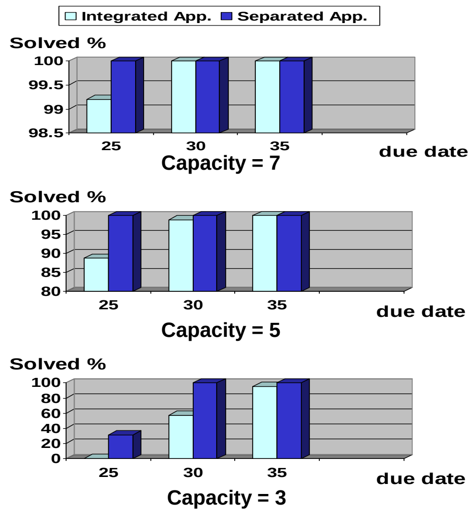


Fig. 13. Comparison on Number of Problem Solved between the Integrated Approach and the Partitioned Approach.

6.1. Integrated vs. Partitioned

From an analysis of Table 5, several observations can be made. Regarding the number of *solved* problems as shown in Figure 13, we observe the following:

Observation 4. When resource capacity constraints are loose, the integrated approach performs comparably to the partitioned scheme under different deadline settings. However, in problems with tighter capacity constraints we find that the partitioned approach solves many more problems.

With regard to *quality*, the results are more surprising. As shown in Figure 14, we observe:

Observation 5. Surprisingly, even though the integrated scheme takes into account the quality goal at each step of the solving process, it doesn't show an advantage in producing higher quality. Instead, it turns out to produce lower-quality solutions on most of data sets. In particular, it performs worse on hardest problems.

Table 5

Experimental results

	due date = 25			due date = 30			due date = 35		
	solved	**constr.**	**quality**	**solved**	**constr.**	**quality**	**solved**	**constr.**	**quality**
Integrated Approach									
C=3	0.8%	-	-	57.0%	62.03	48.51%	95%	64.44	48.67%
C=5	88.8%	23.66	80.75%	98.8%	24.67	79.91%	100%	24.91	79.68%
C=7	99.2%	6.11	95.45%	100%	6.07	95.33%	100%	6.09	95.37%
Partitioned Approach									
C=3	31.2%	14.30	47.21%	100%	13.94	50.87%	100%	13.93	52.01%
C=5	100%	6.33	81.60%	100%	6.27	81.37%	100%	6.32	81.19%
C=7	100%	2.51	95.37%	100%	2.52	95.28%	100%	2.54	95.19%

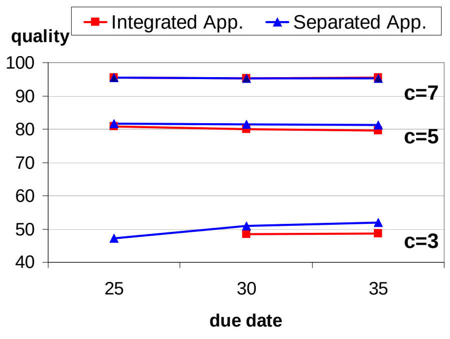


Fig. 14. Comparison on Quality between the Integrated Approach and the Partitioned Approach.

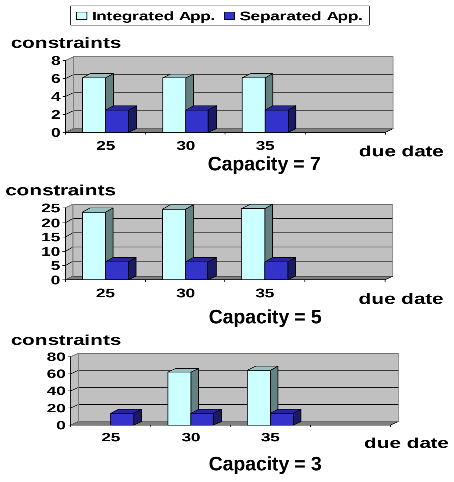


Fig. 15. Comparison on Number of Posted Constraints between the Integrated Approach and the Partitioned Approach.

This behavior is confirmed by the number of posted constraints reported in Figure 15, where we can see that the partitioned scheme posts fewer constraints than the integrated approach. Practically, a schedule with fewer constraints is more flexible. Therefore, the above results further confirm that there is a synergy between the objective of maintaining temporal flexibility and that of maximizing quality, as we found in the previous section. The partitioned scheme builds up a more flexible schedule at the first stage. This flexibility helps to produce better high-quality schedules in the second stage. By attempting to consider quality maximization along the way to generating a resource feasible solution, the integrated approach lessens its opportunities to retain temporal flexibility.

6.2. *A Further Analysis of Temporal Flexibility*

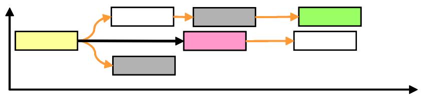
To further confirm our hypothesis about the utility of maintaining temporal flexibility from the standpoint of maximizing quality, we replace the iterative chaining procedure used in our previous experiments with *Simple Chaining*. In *Simple Chaining*, the $SelectChain(a_i)$ sub-procedure always dispatches the next activity a_i to the first available resource unit (chain). Table 6 compares the results obtained using these two chaining versions. We can see that the solution quality obtained with *simple chaining* degrades greatly compared to the algorithm with *iterative chaining*. In particular in the case of $C = 7$, the quality values are reduced from approximately 95% to approximately 70%.

We illustrate the importance of temporal flexibility in the partitioned scheme approach by a simple example shown in Figure 16. Figure 16(a) presents a different $\mathcal{POS}$ for the same fixed times schedule in Figure 9(b). It has greater flexibility than the $\mathcal{POS}$ in Figure 9(c) (as there are fewer constraints between pairs of chains) and thus is capable of producing a solution of better quality (compare Figure 9(d) and Figure 16(b)).

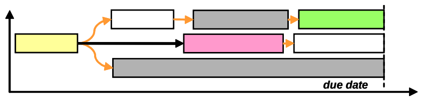
Table 6

Quality values using *simple chaining* vs. *iterative chaining*

quality	due date = 25	due date = 30	due date = 35
Simple Chaining			
C=3	46.39%	49.03%	49.88%
C=5	67.53%	67.67%	68.26%
C=7	70.03%	70.57%	71.55%
Iterative Chaining			
C=3	47.21%	50.87%	52.01%
C=5	81.60%	81.37%	81.19%
C=7	95.37%	95.28%	95.19%



(a) Alternative $\mathcal{POS}$ w.r.t. Figure 9(c)



(b) Better solution w.r.t. Figure 9(d)

Fig. 16. The effects of temporal flexibility on the quality solution.

7. Conclusions

In this paper, we have investigated a *new* type of scheduling problem for discretionary tasks, whose duration is not an inherent property of tasks, but instead a decision to be made by the scheduler: The longer a task is allowed to execute, the higher the quality of its result. The management goal is to maximize the cumulative quality of all tasks subject to temporal and resource constraints. This additional degree of freedom for decision-making in precedence and resource constrained scheduling environments introduces a new challenge for scheduling research.

Major contributions of this paper are as follows. First, we design a polynomial algorithm for generating the optimal schedule in the single capacity case and we prove the general (multiple capacity) version of this problem is NP-complete. Second, we present two new precedence constraint posting algorithms for solving this general problem, along with several search control heuristics for resolving resource conflicts. Our solution procedures incorporate a linear program as a subprocedure for optimally setting activity durations into the search for a resource feasible schedule with high quality. Our experimental results confirm the overall effectiveness of our approaches.

The third contribution is in understanding the key considerations that underlie the design of effective search control heuristics and the hybrid framework integrating constraint-posting and linear optimization:

- To successfully find a schedule with high quality, a good tradeoff must be made between minimizing the quality loss as each new constraint is posted, and retaining temporal flexibility to accommodate additional precedence constraints on subsequent cycles.
- Comparison on the two solution procedures indicates a simple and effective idea that temporally flexible solutions provide better *grist* for generating higher quality schedules. This potential synergy between objectives of maintaining temporal flexibility and maximizing quality implies that many existing techniques in building flexible schedules can be adapted to solve this new class of problems.

Appendix: Proof of Theorem 1

First, we prove the **optimality of the algorithm for the single capacity problem**. There are three cases of the minimum duration schedule resulting from the greedy approach in Step 2 in the algorithm.

- Case 1. The end time of the last activity has passed deadline, then there is no feasible solution. This is because in the single capacity problem, the greedy approach generates a minimum makespan schedule. If this schedule cannot meet the deadline, no schedule can meet it.
- Case 2. There is no idle period in the schedule, then the resulting minimum duration schedule is the optimal schedule and the algorithm stops.
- Case 3. There are some idle periods in the schedule.

Results in Case 1 and Case 2 are obvious. We only need to prove proceeding from the minimum duration schedule in Case 3, the algorithm can find an optimal solution .

We use the notation $Q(S)$ to represent the quality of a schedule S. Let S^* be the schedule resulting from the algorithm and S be an arbitrary schedule. Then we want to prove $Q(S^*) \geq Q(S)$.

Assuming the minimum duration schedule resulting from Step 2 is S_0. In S_0, we denote a_{j_1} as the activity with the maximum slope and p_1 as the end time of the

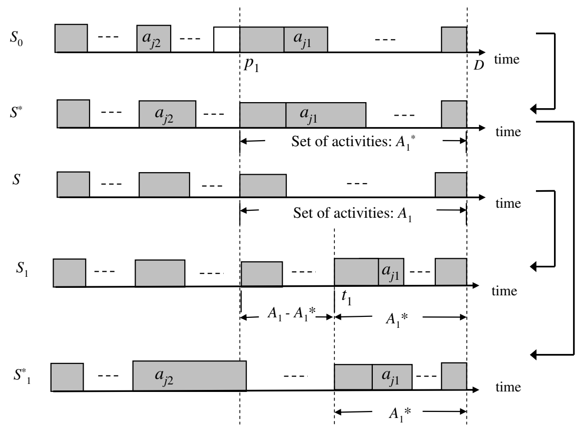


Fig. 17. Schedules in the Proof of the Single Capacity Algorithm

first idle period before a_{j_1}. If there is no idle period before a_{j_1}, let $p_1 = \min\{r_i\}$, and without loss of generality, we can assume $\min\{r_i\} = 0$. Then according to Step 2 in the algorithm, any activity scheduled later than p_1 in S_0 cannot start before p_1 in any schedule. Note that this statement remains true for S^* after Step 3 and 4. Please refer to Figure 17 for the schedules mentioned in this proof.

Let A_1^* and A_1 be the sets of activities scheduled later than p_1 in S^* and S respectively, then $A_1^* \subseteq A_1$ because A_1 may contain the activities which can start before p_1. The set of these activities is $A_1 - A_1^*$.

Next we construct two new feasible schedules S_1^* and S_1 from schedules S^* and S, and two subschedules L_1^* and L_1 in S_1^* and S_1 respectively.

(i) Constructing S_1 from S

In S, we rearrange activities such that all the activities in $A_1 - A_1^*$ are scheduled to start before all the activities in A_1^*, reserving their old orderings. The resulting schedule S_1 is still feasible and

$$Q(S_1) = Q(S). \tag{12}$$

In S_1, the latest end time of activities in $A_1 - A_1^*$ is denoted as t_1, as shown in Figure 17.

(ii) Constructing S_1^* from S^*

In S^*, we denote a_{j_2} as the activity of the maximum slope among activities before p_1. We increase the end time of a_{j_2} by $t_1 - p_1$ and shrink the duration of a_{j_1} by $t_1 - p_1$ such that the start time of the first activity in A_1^* is t_1. The resulting schedule S_1^* is feasible and has an equal or lower quality than S^*:

$$Q(S^*) \geq Q(S_1^*). \tag{13}$$

(iii) Comparing L_1^* and L_1, two subschedules within $[t_1, D]$ in S_1^* and S_1

Because the two subschedules contain the same set of activities, A_1^* and according to the algorithm, the activity with the maximum slope, a_{j_1}, is stretched the most in L_1^*, we obtain

$$Q(L_1^*) \geq Q(L_1). \tag{14}$$

From Equation (12)-(13), in order to prove $Q(S^*) \geq Q(S)$, we need only to prove $Q(S_1^*) \geq Q(S_1)$. If there is no idle period in the subschedules within $[0, t_1]$ in S_0, then $p_1 = t_1 = 0$, and $L_1^* = S_1^*$, $L_1 = S_1$, and we have

$$Q(S^*) \geq Q(S).$$

Thus, we need only to compare the subschedules within $[0, t_1]$ in S_1^* and S_1, when there are some idle periods in the subschedules within $[0, t_1]$ in S_0. The subschedules within $[0, t_1]$ in S_1^* and S_1 can be considered as two new schedules with a new project deadline t_1, and we can repeat the above procedure on these two subschedules while keeping L_1^* and L_1 unchanged. Then we can construct two new feasible schedules S_2^* and S_2 from schedules S_1^* and S_1, and two subschedules L_2^* and L_2 within $[t_2, t_1]$ in S_2^* and S_2 respectively, satisfying

$$\begin{cases} Q(S_2) = Q(S_1), \\ Q(S_1^*) \geq Q(S_2^*), \\ Q(L_2^*) \geq Q(L_2). \end{cases} \tag{15}$$

Actually we can repeat the above procedure (i), (ii) and (iii) when there are some idle periods in the subschedules within $[0, t_i]$ in S_0: we can construct two new feasible schedules S_{i+1}^* and S_{i+1} from schedules S_i^* and S_i, and two subschedules L_{i+1}^* and L_{i+1} within $[t_{i+1}, t_i]$ in S_{i+1}^* and S_{i+1} respectively, and obtain that

$$\begin{cases} Q(S_{i+1}) = Q(S_i), \\ Q(S_i^*) \geq Q(S_{i+1}^*), \\ Q(L_{i+1}^*) \geq Q(L_{i+1}). \end{cases} \tag{16}$$

Denote by m the above maximum steps. Then there is no idle period in the subschedules within $[0, t_m]$ in S_0, and the subschedules K_m^* and K_m within $[0, t_m]$

in S_m^* and S_m are the same, that is $K_m^* = K_m$, and we have

$$\begin{cases} Q(S_m^*) = Q(K_m^*) + \sum_{i=1}^{m} Q(L_i^*), \\ Q(S_m) = Q(K_m) + \sum_{i=1}^{m} Q(L_i). \end{cases} \tag{17}$$

From Equation (12)-(17), we conclude that

$$Q(S^*) \geq Q(S_m^*) \geq Q(S_m) = Q(S),$$

that is, the schedule S^* resulting from the algorithm is an optimal solution.

Second, we prove **the complexity of the single capacity algorithm**.

Assume n is the number of activities in the problem. Without loss of generality, we assume all the release dates have been propagated according to the precedence constraints, which means: if activity$i$ and activity$j$'s release dates are r_i and r_j, and $(i, j) \in E$, we update j's release date as $max\{r_j, r_i + d_i\}$. Then $r_i \leq r_j$, if $(i, j) \in E$.

In the step of building a minimum duration schedule, creating an increasing order of release dates of all the activities takes time $O(nlogn)$. Then we work with the activities one by one in order of increasing release dates. If an activity is eligible, allocate it into the schedule, then update the eligibility of other activities, which takes $O(n)$. Continue to do this until all the activities have been scheduled. So finding a greedy solution assuming activities run for their minimum duration needs time $O(n^2)$.

In the step of filling the idle times, we need to create a decreasing order of all the activities according to their slopes of quality functions, which takes time $O(nlogn)$. Before we fill in the first idle period, we search from the head of the ordering, and stop when we find the activity which is positioned before this idle period. At this moment, we already know the first n_1 activities in that ordering are positioned after this idle period. Because they won't be selected in the next steps, we delete the n_1 activities from the slope ordering and stretch the selected activity to fill in the idle period. Then we do the same thing with the second idle period and the $n - n_1$ activities in the remaining slope ordering. After all the idle periods are filled in, the total number of searched activities is at most n. So this step takes O(n+nlogn).

Therefore the complexity will be $O(n^2)$.

Third, we prove **the complexity of the multiple capacity problem**.

We prove the complexity of the multiple capacity problem by reducing a known *NP*-completeness instance to this problem.

Definition 3. Multiple Capacity Single Resource Quality Sum Problem with Linear Quality Profiles We are given a set A of activities, each activity i having duration not shorter than $d_i \in Z^+$, number $C \in Z^+$ units of resource, partial order $\prec$ on A, for each activity $i \in A$ a release date $r_i \in Z^+$ and common deadline $D \in Z^+$. Can we decide the start time s_i' and end time e_i' for each activity i to maximize the linear quality sum, obey the precedence constraints and meets all the deadlines?

Definition 4. Multiprocessor Scheduling with Individual Deadlines We are given a set T of tasks, each task i having length $l_i = 1$, number $m \in Z^+$ of processors, partial order $\prec$ on T, for each task $t \in T$ a deadline $D_i \in Z^+$ and a common release date zero. Is there a m-processor schedule σ for T that obeys the precedence constraints and meet all the deadlines?

Multiprocessor Scheduling with Individual Deadlines is known to be *NP*-complete [16].

This optimization problem is harder than the problem of finding a feasible solution satisfying all the constraints: precedence, minimum duration, resource capacity, individual release dates and common deadline. So in order to prove the original problem is *NP-complete*, we only need to prove the problem of finding a feasible solution is *NP-complete*. According to [16], the proof has two steps: First, we prove finding a feasible solution is *in NP*; second, we prove it is *NP-hard*.

The problem of finding a feasible solution is *in NP* because the following verifier for this problem runs in polynomial time in the number of activities n:

Given a set A of activities, if we have the values for the start time s_i' and the end time e_i' of activity i.

- Checking minimum duration constraint $e_i' - s_i' \geq d_i$ needs $O(n)$ time.
- Checking precedence constraint needs $O(n^2)$ time.
- Checking the common deadline constraint $e_i' \leq D$ needs $O(n)$ time.
- Checking individual release date constraint $s_i' \geq r_i'$ needs $O(n)$ time.

To show *NP*-hardness, we reduce an arbitrary instance of multiprocessor scheduling with individual deadlines into the following instance of our problem.

Each activity in A corresponds to each task in T, the precedence direction in A is the opposite direction of the partial order in T. Let $d_i = 1$, $D = max_i(D_i)$, $r_i = D - D_i$.

Then if multiprocessor problem's instance has a feasible solution (s_i, e_i), we get $e'_i = D - s_i$ and $s'_i = D - e_i$ are feasible for the above instance of our problem.

- Check minimum duration constraint, $e'_i - s'_i = e_i - s_i = 1 \geq d_i$;
- Check precedence constraint, $s'_i - e'_j = s_j - e_i \geq 0$, which is because for any $(j, i) \in E(A)$, we have $(i, j) \in E(T)$;
- Check the common deadline constraint, $e'_i = D - s_i \leq D$;
- Check individual release date constraint, $s'_i = D - e_i \geq D - D_i = r_i$.

On the other hand, if the above instance of our problem has a feasible solution (s'_i, e'_i), we change the solution into (s''_i, e'_i), where s'_i is increased to s''_i in order to make the duration equal to 1. (s''_i, e'_i) is still feasible for our problem instance. Then, we get $e_i = D - s''_i$ and $s_i = D - e'_i$ are feasible for the multiprocessor problem's instance.

- Check duration constraint, $e_i - s_i = e'_i - s''_i = 1$;
- Check partial order constraint, $s_j - e_i = s''_i - e'_j \geq 0$, which is because for any $(i, j) \in E(T)$, we have $(j, i) \in E(A)$;
- Check the individual deadline constraint, $e_i = D - s''_i \leq D - r_i = D_i$;
- Check common release date constraint, $s_i = D - e'_i \geq 0$.

References

[1] Ajili, F., H. El Sakkout. 2003. A Probe-Based Algorithm for Piecewise Linear Optimization in Scheduling. *Annals of Operations Research* **118** 35–48.

[2] Anand, Krishnan, M. Fazil Pac, Senthil K. Veeraraghavan. 2008. Quality-speed conundrum: Tradeoffs in labor-intensive services. Wharton School Working Paper. University of Pennsylvania.

[3] Baptiste, P., C. Le Pape, W. Nuijten. 2001. *Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, Norwell, MA.

[4] Beck, J.C. 1999. Texture Measurements as a Basis for Heuristic Commitment Techniques in Constraint-Directed Scheduling. Ph.D. thesis, Dept. of Computer Science, University of Toronto.

[5] Beringer, Hernri, Bruno De Backer. 1995. *Logic Programming: Formal Methods and Practical Applications*, chap. Combinatorial Problem Solving in Constraint Logic Programming with Cooperating Solvers. Elsevier Science, Amsterdam, Holland.

[6] Cesta, A., A. Oddi, S. F. Smith. 1998. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. *Proceedings of the 4^{th} International Conference on Artificial Intelligence Planning Systems, AIPS-98*. 214–223.

[7] Cesta, A., A. Oddi, S. F. Smith. 1998. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. *In Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems,AIPS-98*. 214–223.

[8] Cesta, A., A. Oddi, S. F. Smith. 2000. Iterative Flattening: A Scalable Method for Solving Multi-Capacity Scheduling Problems. *Proceedings of the 17^{th} National Conference in Artificial Intelligence (AAAI'00)*. Austin, TX.

[9] Cesta, A., A. Oddi, S. F. Smith. 2002. A Constraint-Based Method for Project Scheduling with Time Windows. *Journal of Heuristics* **8**(1) 109–136.

[10] Cheng, C., S.F. Smith. 1996. A Constraint Satisfaction Approach to Makespan Scheduling. *Proceedings 4th International Conference on Artificial Intelligence Planning Systems*.

[11] Dean, T. L., M. Boddy. 1988. An analysis of time-dependent planning. *Proceedings of the 7^{th} National Conference on Artificial Intelligence*. AAAI Press, St. Paul, MN, 49–54.

[12] El Sakkout, H., M. Wallace, T. Richards. 1998. Minimal Perturbation in Dynamic Scheduling. *Proc. of the 13th European Conference on Artificial Intelligence (ECAI-98)*.

[13] El Sakkout, H. H., M. G. Wallace. 2000. Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *Constraints, Special Issue on Industrial Constraint-Directed Scheduling* **5**(4).

[14] Fulkerson, D.R. 1961. A Network Flow Computation for Project Cost Curves. *Management Science* **7**(2) 167–178.

[15] Gans, Noah, Ger Koole, Avishai Mandelbaum. 2003. Telephone call centers: Tutorial, review, and research prospects. *Manufacturing Service Oper. Management* **5**(2) 79–141.

[16] Garey, M.R., D.S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman and Company, San Francisco, California.

[17] Hooker, John. N. 2004. A hybrid method for planning and scheduling. *Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*. Toronto, Canada.

[18] Hopp, Wallace J., Seyed M.R. Iravani, Fang Liu. 2006. White collar workforce management: An operations-oriented survey. *Production and Operations Management, To Appear* .

[19] Hopp, Wallace J., Seyed M.R. Iravani, Gigi Y. Yuen. 2007. Operations systems with discretionary task completion. *Management Science* **53**(1) 61–77.

[20] Horvitz, E. 1987. Reasoning about beliefs and actions under computational resource constraints. *Third Workshop on Uncertainty in Artificial Intelligence*. Seattle, Washington.

[21] Icmeli, Oya, S.Selcuk Erenguc. 1996. The Resource Constrained Time-Cost Tradeoff Project Scheduling Problem with Discounted Cash Flows. *Journal of Operations Management* **14** 255–275.

[22] Kelley Jr., J.E. 1961. Critical Path Planning and Scheduling: Mathematical Basis. *Operations Research* **9**(3) 296–320.

[23] Kelley Jr., J.E., M.R. Walker. 1959. *Critical Path Planning and Scheduling: An introduction*. Mauchly Associates Inc.

[24] Khatib, L., P. H. Morris, R. A. Morris, F. Rossi. 2001. Temporal Constraint Reasoning With Preference. *Proceedings of the* 17^{th} *International Joint Conference on Artificial Intelligence*.

[25] Nuijten, W.P.M., E.H.L. Aarts. 1994. Constraint Satisfaction for Multiple Capacitated Job-shop Scheduling. *Proceedings of the 11th European Conference on Artificial Intelligence*.

[26] Oddi, A., A. Cesta. 1997. A Tabu Search Strategy to Solve Scheduling Problems with Deadlines and Complex Metric Constraints. *Proceedings of the 4th European Conference on Planning*.

[27] Philips, S., M.I. Dessouky. 1977. Solving the Project Time-Cost Tradeoff Problem Using the Minimal Cut Concept. *Management Science* **24**(4) 393–400.

[28] Policella, N., A. Oddi, S. F. Smith, A. Cesta. 2004. Generating Robust Partial Order Schedules. M. Wallace, ed., *Principles and Practice of Constraint Programming,* 10^{th} *International Conference, CP 2004*, *Lecture Notes in Computer Science*, vol. 3258. Springer, 496–511.

[29] Policella, N., S. F. Smith, A. Cesta, A. Oddi. 2004. Generating Robust Schedules through Temporal Flexibility. *Proceedings of the* 14^{th} *International Conference on Automated Planning & Scheduling, ICAPS'04*. AAAI, 209–218.

[30] Policella, Nicola, Amedeo Cesta, Angelo Oddi, Stephen F. Smith. 2007. From Precedence Constraint Posting to Partial Order Schedules. A CSP approach to Robust Scheduling. *AI Communications* **20**(3) 163–180.

[31] Policella, Nicola, Amedeo Cesta, Angelo Oddi, Stephen F. Smith. 2009. Solve-and-Robustify. Synthesizing Partial Order Schedules by Chaining. *Journal of Scheduling* **12**(3) 299–314.

[32] Policella, Nicola, Xiaofang Wang, Stephen F. Smith, Angelo Oddi. 2005. Exploiting Temporal Flexibility to Obtain High Quality Schedules. *Proceedings of the* 20^{th} *National Conference on Artificial Intelligence, AAAI-05*.

[33] Russel, S., E.H. Wefald. 1991. *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge,MA.

[34] Sadeh, N. 1991. Look-Ahead techniques for Micro-Opportunistic Job Shop Scheduling. Ph.D. thesis, Dept. of Computer Science, Carnegie Mellon University.

[35] Slowinski, R. 1980. Two Approaches to Problems of Resource Allocation Among Project Activities-A Comparative Study. *J.Operational Research Society* **31**(8) 711–723.

[36] Slowinski, R. 1981. Multiobjective Network Scheduling with Efficient Use of Renewable and Non-renewable Resources. *European J.Operational Research* **7**(3) 711–723.

[37] Smith, S. F., C. Cheng. 1993. Slack-Based Heuristics for Constraint Satisfaction Scheduling. *Proc.11th National Conference on Artificial Intelligence*. AAAI Press, Wash DC, 139–144.

[38] Talbot, F. B. 1982. Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case. *Management Science* **28**(10) 1197–1210.

[39] Wang, Xiaofang, Stephen F. Smith. 2005. Retaining Flexibility to Maximize Quality: When the Scheduler Has the Right to Decide Activity Durations. *Proceedings of the 15*th *International Conference on Automated Planning & Scheduling, ICAPS'05*.

[40] Zilberstein, S. 1993. Operational rationality through compilation of anytime algorithms. Ph.D. thesis, Computer Science Division, University of California at Berkeley.