Generating Schedules to Maximize Quality

Xiaofang Wang[1] and Stephen F. Smith[2]

[1] Tepper School of Business, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh,PA 15213
xiaofanw@andrew.cmu.edu
[2] Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh,PA 15213
sfs@cs.cmu.edu

Abstract. In knowledge generating production processes such as intelligence gathering and news reporting, the quality of the result produced by a given activity depends on its duration, and, due to resource limitations and process deadlines, tradeoffs must invariably be made regarding how much time to devote to various activities to achieve maximum overall effect. In essense, some activities must be executed in faster time cycles than would be desirable under non-constraining circumstances, with consequent degradation of process quality. In this paper, we consider this type of scheduling problem, which we refer to generally as quality maximization. Starting with normal resource-constrained project scheduling problem (RCPSP) assumptions, we define a new type of scheduling problem by additionally associating a quality profile with each activity in the project. Quality profiles have an "anytime" property, implying that activities can be terminated at any point, with the quality of the output being proportional to duration. Instead of finishing all activities as fast as possible, the goal of scheduling in this context is to maximize the overall quality given a hard project due date. We formulate this quality maximization problem as a constraint-based optimization problem, and present a new constraint posting algorithm for solving this problem that incorporates a linear optimization program. Different constraint posting heuristics are defined and evaluated on a set of quality maximization RCPSP problems constructed from standard reference RCPSP problems. The experimental results show the overall effectiveness of our approach for generating schedules to maximize quality. And ratio-based heuristics provide a promising starting point for stochastic sampling or other schedule refining techniques by solving 100% problems without backtracking.

1 Introduction

In the past few years, constraint satisfaction problem solving (CSP) techniques have been successfully applied to several classes of scheduling problems[1–8]. Most of this work has focused on generating feasible schedules, or on optimizing some traditional scheduling performance measure such as makespan or weighted tardiness. Another important, but typically ignored, performance objective in many practical domains is that of maximizing the *quality* of the outputs (or products) of scheduled processes. Quality maximization scheduling problems are particularly prevalent in so-called knowledge-intensive production processes, where knowledge products are created and manipulated by shared (typically human) resources to meet demands within tight deadlines. Such processes exist within such settings as news reporting, intelligence gathering, and new product research and development.

Consider the process of creating an issue of a weekly news magazine. Several different types of activities need to take place: information gathering, data analysis, writing, editing, confirming and duplicating sources, responding to late-breaking events, etc. Activities have causal inter-dependencies and in many cases are also competing for the same resources. For example, the writing of a news story must precede any editing, and the fact that multiple news events are being covered will create contention for available news personnel. The quality of many activities will be a function of the time that can be devoted to it: a story written in an hour by a news correspondent will be lower quality than it would be if a day was spent on it. Given the production deadline associated with publishing the next issue, decisions will invariably need to be made regarding which activities to spend more time on, and the best decisions are those that maximize the quality of the final content of the issue.

Such a production process can be seen generally as a project, and can be modelled as an extended resource constrained project scheduling problem (RCPSP)[9]. In addition to the normal set of constraints included in the definition of RCPSP, a specification of each activity's expected productivity is also incorporated. Analogous to the concept of performance profiles in anytime algorithms, a quality profile is associated with each activity, which 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 perhaps to a required minimum duration), with an output quality proportional to its duration. We refer to this extended RCPSP as the Quality Maximization RCPSP (QM-RCPSP). Whereas under standard RCPSP formulations, the goal is to establish resource-feasible activity start (end) times that optimize make-span or on-time performance assuming fixed durations, the objective in the QM-RCPSP is to determine activity durations that maximize overall output quality given project release and deadline constraints. In the simplest case, overall output quality is defined as the sum of the output quality of each individual activity.

In this paper, we propose a precedence constraint posting (PCP) procedure for solving this quality maximization scheduling problem and investigate initial constraint posting heuristics for producing good quality solutions. Our approach draws on two complementary results from previous research. In [10] a PCP procedure for iteratively transforming a time-feasible solution into a resource feasible solution by successive "levelling" of resource conflicts is shown to provide an effective basis for solving difficult instances of RCPSP/Max (an extended version of the RCPSP formulation considered here that additionally incorporates minimum and maximum time lags between pairs of activities). Given this result and the related nature of QM-RCPSP, we adopt this basic schedule generation framework. A second element of our approach is drawn from more classic project management research on the time/cost tradeoff problem[11], which addresses the question of whether activity durations should be reduced (by using more resources) at some increase in cost. It has been shown[12–14] that under the assumption of linear and continuous cost function for each activity, the uncapacitated version of the time/cost tradeoff problem can be efficiently solved to optimality. Noting the similarity of this computation to the computation required to compute maximum quality durations in a project network, we utilize this procedure as the "constraint propagation" step of the solution procedure, to adjust activity durations when a new precedence constraint is posted.

The remainder of the paper is organized as follows. In Section 2, we formulate the quality maximization problem and characterize its complexity. Then, in Section 3, we present our constraint posting algorithm and define a set of constraint-posting heuristics. In Section 4 we report the experimental results that indicate the relative performance of various heuristics and give evidence of the overall effectiveness of our approach. Related work is briefly summarized in Section 5. Finally, in Section 6, some conclusions are drawn.

2 The Quality Maximization Resource Constrained Project Scheduling Problem(QM-RCPSP)

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 resources with limited capacity, the quality maximization scheduling 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. 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 i,
- e_i: decision variable, the end time of activity 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[3] 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.[4]

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

[4] Our model can easily be extended to the problems where each activity or task can require more than one unit of resource. In that case, we can divide an activity into

– Q: objective, the total quality output from the project.

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

$$A(S,t) := \{i \in V \mid s_i \leq t < e_i\}(t \geq 0)$$

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. So the *resource constraint* is

$$R(S,t) \leq C$$

Then the problem 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

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

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

$$d_i \leq e_i - s_i, i \in V, \tag{4}$$

$$r_i \leq s_i, i \in V, \tag{5}$$

$$e_i \leq D, i \in V, \tag{6}$$

(2), (3), (4), (5) and (6) are precedence, resource, minimum duration, release date and due date constraints respectively.

2.1 Complexity Analysis

For the single capacity ($C = 1$) problem, there exist a polynomial algorithm to solve it optimally.The main idea is to greedily schedule all activities assuming their minimum duration and expand some of the activities in the best way to fill in all the idle periods.[5]

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.

[5] an idle period is the period of time in which no activities are running; oppositely, the period of time in which one 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.

```
Algorithm for Single Capacity Problem
Input:A set of activities and the constraints.
Output: An optimal schedule.
1. Output:use a greedy approach to schedule all activities assuming their minimum duration:
2.   start at earliest release date
3.   schedule an activity from a set of "eligible" activities at current time t: i.e.
       (1) the activity whose release date is reached
       (2) all of its predecessors have been finished
4. while there are some idle periods in the current schedule, start backward from the due-date
5.   for each idle period:
6.     find the activity who has the maximum quality slope and is before the idle period
7.     increase its duration until this idle period shrinks to zero
8. Return current schedule
```

Fig. 1. Algorithm for Single Capacity Problem

The complexity of this algorithm is $O(n^2)$. Please refer to appendix A for the proof of correctness and complexity analysis.

However, the multiple capacity ($C > 1$) problem can be proved to be *NP*-complete. Please refer to appendix B for the proof of complexity.

3 A Constraint Posting Algorithm for Solving QM-RCPSP

We adopt a precedence constraint posting (PCP) approach to solve our quality maximization scheduling problem (QM-RCPSP), motivated by the effective prior use of this type of algorithm in standard RCPSP contexts[10]. Since our goal is to maximize quality, we adapt the traditional PCP framework to take advantage of an optimization procedure for solving the related time/cost tradeoff problem[11]. We first define our basic PCP procedure, and then propose several constraint posting heuristics.

3.1 The Precedence Constraint Posting Framework

Following the tradition in previous PCP work, we begin by defining the notion of a contention peak.

Definition 1. *Given a schedule, a* ***contention peak*** *(or simply a peak) is a set of activities, each of which simultaneously requires one unit of resource and whose total resource requirement exceeds the resource capacity in the time window* $[t_1, t_2]$*, the* ***length*** *of a peak is* $(t_2 - t_1)$.

As usually formulated, a *PCP* scheduling procedure acts to transform a time feasible solution into a resource feasible solution by eliminating all contention peaks. At each step of the search, a peak is selected for “levelling”, and a precedence constraint is posted between some pair of activities in the peak to achieve this effect. Each time a constraint is posted, constraint propagation is performed, to compute new activity start and completion times and confirm that the solution is still feasible.

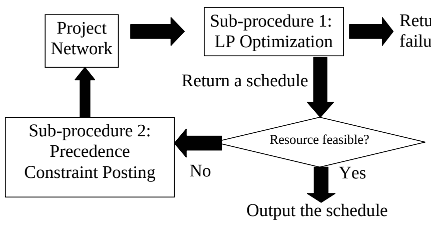

Fig. 2. The Precedence Constraint Posting Framework

In the case of QM-RCPSP, 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. In brief, the time-cost tradeoff problem is a classical variant of RCPSP which considers the option of speeding up an activity by spending more on it. The goal is to meet the project's due date, while minimizing total crashing costs. If the cost function for each activity is linear and continuous, the problem can be optimally solved in polynomial time by linear programming. [12–14].

Given this result, we can reformulate the *PCP* procedure to be one of transforming an initial "quality optimized" initial solution into a resource-feasible solution that optimizes quality through elimination of contention peaks. Essentially this corresponds to replacing the traditional constraint propagation step with a corresponding call to an LP Solver to solve the time/cost tradeoff problem for the current activity precedence graph. If the LP solver returns a solution, it will be the optimal quality solution for this graph. If it returns failure, then the search has reached an infeasible state. The PCP framework is illustrated in Fig. 2 and the algorithm procedure is described in Fig. 3.

3.2 Constraint-posting Heuristics

We propose two classes of constraint-posting heuristics: *Peak-Driven heuristics* and *Quality Loss Look-ahead Heuristics.* Following traditional formulations of PCP scheduling procedures, Peak Driven Heuristics first select a peak based on some feature from the peak and then determine a pair of activities in the peak to sequence. Quality loss look-ahead heuristics, alternatively look at all conflicting activities in current schedule, select the activity most likely to lead to minimal degradation of quality, and then look for a second competing activity to sequence.

```
Heuristic-based Constraint Posting Algorithm
Input: A set of activities with constraints.
Output: A solution or failure
1.  loop
2.    apply linear programming 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
```

Fig. 3. Heuristic-based Constraint Posting Algorithm

Peak Driven Heuristics Observing that an activity's length is closely related to the quality contribution from itself and the other activities running in parallel with it, **peak length** would seem to be a reasonable peak selection criterion. We consider three possibilities: longest peak, shortest peak and random (length doesn't matter).

After choosing a peak, we intuitively prefer to post constraints between activities in this peak with smallest quality **slopes** (k_i) because shrinking these activities would seem to lead to the smallest quality loss. But let us consider an activity already at its minimum duration. It can not be shrunk at all, and from its slope we can't say anything about quality loss. Given this observation, we define a second activity selection heuristic by introducing duration into the activity selection criterion. Specifically we choose the activities with the largest **ratio** of "reducible duration" to slope($\frac{e_i - s_i - d_i}{k_i}$), where the reducible duration of an activity is the amount greater than its minimum duration. From this choice, we actually bias on the choice toward activities with small slope and long reducible duration. In all cases, the two selected activities are sequenced according to the rule of Earliest Start Time First. Altogether, we list the possible heuristics in Table 1.

Quality Loss Look-ahead Heuristics This set of heuristics focuses directly on posting the constraint that leads to minimal quality loss. The first activity is selected from some peak according to either the criterion of slope or ratio. The second activity selection and sequencing decision are based on one criterion—**quality loss**. Quality loss is the difference of the two optimal[6] quality values

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

Table 1. Peak Driven Heuristics

Heuristics	Peak Selection	Activity Selection in the peak	Sequence
LPF-slope	LongestPeakFirst	SmallestSlopes	EarliestStartTimeFirst
RPF-slope	RandomPeakFirst	SmallestSlopes	EarliestStartTimeFirst
SPF-slope	ShortestPeakFirst	SmallestSlopes	EarliestStartTimeFirst
LPF-ratio	LongestPeakFirst	LargestRatios	EarliestStartTimeFirst
RPF-ratio	RandomPeakFirst	LargestRatios	EarliestStartTimeFirst
SPF-ratio	ShortestPeakFirst	LargestRatios	EarliestStartTimeFirst

from the problems with and without the candidate precedence constraint respectively. The optimal quality will remain unchanged or degrade after posting one constraint. We try to minimize the degradation—quality loss. To compute it exactly, we can run the linear programming solver(LP) twice (before and after posting a given constraint), but this is very costly in computation time. Hence, we design the following estimation methods.

Quality Loss Estimation Method 1 - 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 overlapping. 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. For different 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 Fig. 4. Because $s_b - s_a < e_a - e_b$, we use the formula in equations (10) and (11). As shown in Fig. 4, $s_b + d_b > s_a$, so the quality loss can be estimated as follows:

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

$$B_{Loss} = k_b * [e_b - s_b - d_b]$$
$$QualityLoss = A_{loss} + B_{Loss}$$

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

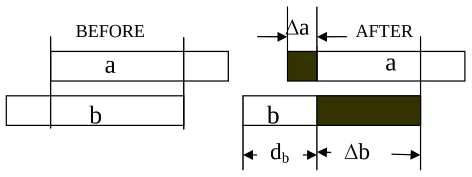

Fig. 4. An Example for Quality Loss Estimation

are generalized in equations(7-11). If the durations of all other activities remain unchanged after re-optimization on the new problem (with the newly posted constraint), the above quality loss estimation will be the actual quality loss. However, in most cases, if we post a new constraint, the other activities will change, too. Thus this method only provide an estimate.

Quality Loss Estimation Method 2 - Gain Only

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 up to Δa, similarly, b's predecessors or successors can grow longer up to Δb. Let the sets of activities which are likely to grow longer be Set_a and Set_b. For example, in Fig. 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 given as follows:

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

We notice that the computation of quality gain is overly optimistic because the activities can be constrained by their other successors or predecessors so that they may not be able to be stretched. However, quality gain also includes the slope information about this activity' predecessors or successors. It is thus actually an indicator of how "connected" this activity is with other activities. Instead of minimizing quality loss, a heuristic that strictly relies on *QualityGain* selects the most connected activity with largest quality gain.

Quality Loss Estimation Method 3 - Loss and Gain If we subtract the quality gain from the above quality loss function, the modified quality loss estimation function is as follows:

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

Altogether, we list the possible heuristics in Table 2.

Table 2. Look-ahead Heuristics

Heuristics	1st Activity Selection	Quality Loss Computation for 2nd Activity Selection
Slope-Loss	SmallestSlope	Estimation1—Loss Only
Slope-Gain	SmallestSlope	Estimation2—Gain Only
Slope-LossGain	SmallestSlope	Estimation3—Loss and Gain
Ratio-Exact	LargestRatio	Exact computation by running LP twice
Ratio-Loss	LargestRatio	Estimation1—Loss Only
Ratio-Gain	LargestRatio	Estimation2—Gain Only
Ratio-LossGain	LargestRatio	Estimation3—Loss and Gain

We should note that the choice of the second activity and how to sequence it with the first activity to minimize quality loss is in fact a myopic decision. When we start with an infinite capacity solution, we actually set an upper bound for the final solution. A good resource levelling heuristic should achieve minimum total quality loss from that upper bound transforming an initial infinite capacity solution into a conflict-free solution. We not only need to minimize the quality loss from posting one constraint, but also need to post as few constraints as possible. Even if the quality loss from posting one constraint is very small, posting too many constraints can reduce quality dramatically.

4 Experimental Evaluation

Because there are no benchmark quality maximization scheduling problems, we adapted a single mode resource constraint project scheduling benchmark problem set[7] to produce a test data set including 400 problems. We used the precedence constraints defined in that problem set. Each problem has 32 activities. And 32 uniformly distributed random integers in the range $[1, 50]$ were generated to represent the slopes for each problem. The algorithms were implemented in Visual C++ 6.0, and LINGO 8.0 was used as the LP solver[8]. The CPU time presented in the following tables were obtained on a Pentium IV-2.40 Ghz processor under Windows XP. We set capacity = 5, due date = 20, minimum duration = 1. Uniformly distributed random integers in the range $[0, 5]$ were generated to represent the release dates for each problem[9]. In the following tables, we compare the performance of heuristics based on following measures:

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

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

[9] We choose this range for release dates in order not to make the temporal constraints too tight guaranteeing that feasible solutions exist for all the problems.

1. *quality(%)*. The average percentage quality[10] normalized to the infinite capacity solution. Infinite capacity solution gives us the upper bound for 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.

Table 3. Results for the 400 problems assuming capacity = 5

	Quality(%)	Runtime(sec.)	Constraints	Solved(%)
LPF-slope	81.6442	1.35	9.29	98.75
RPF-slope	80.1333	1.61	11.07	99
SPF-slope	80.5363	1.20	8.22	98.75
LPF-ratio	77.4414	1.17	8.91	100
RPF-ratio	76.8603	1.15	8.42	100
SPF-ratio	76.8206	1.01	7.36	100
Ratio-Exact	79.4066	59.96	23.33	100
Ratio-Loss	78.9652	2.76	20.18	100
Ratio-Gain	76.5093	1.42	9.73	100
Ratio-LossGain	77.7703	1.56	10.62	100

From the results in Table 3, we can make several observations:

- **solved** Although there is only a slight difference among the heuristics, all ratio-based heuristics solve 100% of problems, while none of slope-based heuristics do. Here is the explanation. If the criterion is slope, a subset of activities will be selected and shrunk again and again due to their small slopes. In the end, they are sequenced and congested on one path in the precedence graph from the first activity to the last activity and all of them are already at or approach minimum durations. Then, if some of them already at minimum durations are selected to be shrunk again, failure is reported. If the criterion is ratio, after the smallest sloped activity is selected and shrunk, it is less likely to be selected in next posting. The activities approaching or at minimum durations are balanced on different paths from the first activity to the last activity. So the ratio-based heuristics tend to avoid infeasibility quite well.
- **quality** Based on the commonly solved problems, slope-based heuristics are better than ratio-based heuristics, although slope-based heuristics solve fewer

[10] Because some heuristics can not solve all the 400 problems, the average percentage quality is calculated based on the commonly solved problems by all heuristics.

problems. This indicates there exists a tradeoff between solution quality and the percentage of solved problems. The reason is slope-based heuristics tend to squeeze small sloped activities together in a chain, which is good to leave space for large sloped activities so as to increase quality, but is fragile to approach infeasibility.
- **runtime** "Ratio-Exact" takes much longer time than others; calling the LP solver to solve time-cost tradeoff problems is the dominant factor on computation cost.
- **constraints** "Ratio-Exact" and "Ratio-Loss" post many more constraints than others.

Note that "Ratio-Exact" does not offer any advantage over the peak-driven heuristics with slopes, which confirms that a completely myopic decision may not be a good choice. To test the heuristics on harder problems, we decease the resource capacity from 5 to 3 to make our problems more severely constrained and rerun the experiments. Because of the huge computational cost, "Ratio-Exact" will be eliminated in the following experiments. Results are given in Table 4.

Table 4. Results for the 400 problems assuming capacity = 3

	Quality(%)	Runtime(sec.)	Constraints	Solved(%)
LPF-slope	54.9231	2.88	21.3	49
RPF-slope	53.374	3.1	22.73	31.25
SPF-slope	53.8113	2.24	16.71	42.75
LPF-ratio	52.4253	2.98	22.01	100
RPF-ratio	51.8315	3.03	22.46	100
SPF-ratio	51.8108	2.63	19.14	100
Ratio-Loss	51.63	5.93	43.34	98
Ratio-Gain	52.9687	3.45	25.49	100
Ratio-LossGain	53.3982	3.7	27.48	100

From the results in Table 4, we can make the following observations:

- **solved** For these more constrained problems, the superiority of ratio-based heuristics becomes clear. Just as we expected, they show a big advantage in avoiding infeasibility. All ratio-based heuristics are much better than slope-based heuristics. Except for "Ratio-Loss", all ratio-related heuristics solve 100%, while the slope-related heuristics solve much fewer problems.
- **quality** For these more constrained problems, "Ratio-Loss" achieves a lower percentage quality than "Ratio-LossGain" although it does better on less constrained problems. This is because "Ratio-Loss" always posts more constraints than "Ratio-LossGain" and the negative influence on quality from

too many redundant constraints is more severe on more constrained problems.

Finally, we examine the effects of an extended stochastic sampling search using the best heuristic found in each group - “LPF-Slope”, “LPF-Ratio” and “Ratio-LossGain”. For each heuristic, the base search procedure is repeatedly restarted some number of times, and the heuristic is used to bias a random choice at each step of the search. We adopt the value-biased sampling approach of [15]. Table 5 indicates the average best results obtained and average cumulative times obtained for each heuristic over 10 iterations. In these experiments, “LPF-Slope(10 iter.)” and “LPF-Ratio(10 iter.)” randomize the peak selection heuristic and bias the selection of contention peak. “Ratio-LossGain(10 iter.)” randomizes the first activity selection and biases the selection on ratio.

Table 5. 10 iterations results for the 400 problems assuming capacity = 3

	Quality(%)	Runtime(sec.)	Constraints	solved(%)
LPF-Slope(10 iter.)	55.8311	30.21	21.88	74.5
LPF-Ratio(10 iter.)	55.1247	27.98	21.31	100
Ratio-LossGain(10 iter.)	55.5087	34.33	25.53	100

Using iterative random sampling, both “Ratio-LossGain” and “LPF-Ratio” achieve comparable percentage quality with “LPF-Slope”. Although the number of problems solvable with “LPF-Slope” is increased significantly, it still shows a big disadvantage in this regard.

5 Related Work

Work on related quality maximization scheduling problems to date has been rather spare. Within the operations research community some work has attempted to bridge the gap between time-cost tradeoff problem solution and scheduling under resource constraints. But this work has restricted attention to either the non-preemptive case with discrete cost functions[16, 17], or the preemptive scheduling case[18, 19]. 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 and continuous cost functions.

From the area of time-bounded computation, Schwarzfischer [20, 21] has recently explored the idea of combining anytime scheduling and deadline scheduling, which he refers to as quality/utility scheduling. Like our model, this work assumes that each activity has an anytime performance profile. However, this work considers the problem of scheduling acyclic task networks on a single processor, whereas our focus is on the multiple capacitated resource problem (corresponding to multiple processors).

6 Conclusion and Extensions

In this paper, we have defined an extended Resource Constrained Project Scheduling Problem(RCPSP)—Quality Maximization RCPSP(QM-RCPSP). We described a new constraint posting algorithm for solving this problem that incorporates a linear program for optimally solving the related time-cost tradeoff problem.

Several heuristics have been proposed and tested. The peak-driven heuristics with simple slope-driven activity selection are found to yield fairly good quality performance for those problem instances that they are able to solve. However, on harder (more-severely constrained) problems, the percentage of problems that are solvable using these simple heuristics degrades significantly. Instead, the ratio based heuristics for activity selection show their big advantage in solving 100% or near 100% of the problems in all cases. This is a promising result because if we can easily get a feasible schedule, many other search techniques can be applied to improve upon this initial schedule; some initial indication of this is given in our experiments using stochastic sampling. These results also indicate that there exists a tradeoff between solution quality and the percentage of solved problems.

Another interesting result is when we try to make a better look-ahead quality loss prediction, we find the prediction that includes an overestimated quality gain can actually achieve comparable percentage of quality while posting much fewer constraints and saving significant computational time. The reason is it actually biases the search toward the more connected activity (the activity with more valuable successors or predecessors). The search benefits from this bias in terms of fewer posted constraints. Just as we see, "Ratio-Loss" always posts more constraints than "Ratio-LossGain" and "Ratio-Gain". When the search is completely guided by the quality gain just as the "Ratio-Gain" does, the solution isn't improved further than "Ratio-LossGain". So we conclude that combining the information from the activities for which we are considering posting a constraint between and their connectivity with other activities achieves a good tradeoff among all the performance measures.

In the future, we will extend our research in the following directions.

- *Multiple projects.* In this paper, we look at scheduling within one project, say, creating one news story. But in a news agency, there are many story projects going on simultaneously with different deadlines and personnel. Extension of our procedure to handle multiple projects is one area of further work.
- *More complex quality dependencies.* In our current model, the only dependencies among activities are precedence relationships. Quality dependencies, which imply that one activity's output may influence it's successors' quality profiles, are more realistic. In that case, instead of a simple summation form, the objective will be a more complex function.
- *Activity Profiles.* We have assumed in this paper that the expected quality of activities can be expressed by linear profiles. To achieve more realistic formulations, linear profiles can be extended to non-linear profiles, and also associated with resources. This latter extension allows for the possibility of differentiating the skill level of different resources.

Acknowledgements. This research was supported in part by the National Science Foundation under contract #9900298 and by the CMU Robotics Institute. The authors thank Anthony Gallagher, Larry Kramer and Terry Zimmerman for helpful comments on an earlier draft.

References

1. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems. Kluwer Academic Publishers, Norwell, MA (2001)
2. Beck, J.: Texture Measurements as a Basis for Heuristic Commitment Techniques in Constraint-Directed Scheduling. PhD thesis, Dept. of Computer Science, University of Toronto (1999)
3. Cesta, A., Oddi, A., Smith, S.: Iterative flattening: A scalable method for solving multi-capacity scheduling problems. In: Proceedings of the Seventeenth National Conference in Artificial Intelligence (AAAI'00), Austin, TX (2000)
4. Cheng, C., Smith, S.: A constraint satisfaction approach to makespan scheduling. In: Proceedings 4th International Conference on Artificial Intelligence Planning Systems. (1996)
5. Nuijten, W., Aarts, E.: Constraint satisfaction for multiple capacitated job-shop scheduling. In: Proceedings of the 11th European Conference on Artificial Intelligence. (1994)
6. Oddi, A., Cesta, A.: A tabu search strategy to solve scheduling problems with deadlines and complex metric constraints. In: Proceedings of the 4th European Conference on Planning. (1997)
7. Sadeh, N.: Look-Ahead techniques for Micro-Opportunistic Job Shop Scheduling. PhD thesis, Dept. of Computer Science, Carnegie Mellon University (1991)
8. Policella, N., Smith, S.F., Cesta, A., Oddi, A.: Generating robust schedules through temporal flexibility. In: Proceedings 14th International Conference on Automated Planning and Scheduling, Whistler, CA (2004)
9. Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operations Research **112** (1999) 3–41
10. Cesta, A., Oddi, A., Smith, S.: A constraint-based method for project scheduling with time windows. Journal of Heuristics **8** (2002) 109–136
11. Kelley Jr., J., Walker, M.: Critical Path Planning and Scheduling: An introduction. Mauchly Associates Inc., Ambler,Pa (1959)
12. Fulkerson, D.: A network flow computation for project cost curves. Management Science **7** (1961) 167–178
13. Kelley Jr., J.: Critical path planning and scheduling: Mathematical basis. Operations Research **9** (1961) 296–320
14. Philips, S., Dessouky, M.: Solving the project time-cost tradeoff problem using the minimal cut concept. Management Science **24** (1977) 393–400
15. Cicirello, V., Smith, S.: Amplification of search performance through randomization of heuristics. In: Proceedings 5th International Conference on Principles of Constraint Programming (CP 02), Ithica, NY (2002)
16. Talbot, F.: Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. Management Science **28** (1982) 1197–1210

17. Icmeli, O., Erenguc, S.: The resource constrained time-cost tradeoff project scheduling problem with discounted cash flows. Journal of Operations Management **14** (1996) 255–275
18. Slowinski, R.: Two approaches to problems of resource allocation among project activities-a comarative study. J.Operational Research Society **31** (1980) 711–723
19. Slowinski, R.: Multiobjective network scheduling with efficient use of renewable and non-renewable resources. European J.Operational Research **7** (1981) 711–723
20. Schwarzfischer, T.: Quality and utility-towards a generalization of deadline and anytime scheduling. In: Proceedings of the 13th International Conference on Automated Planning and Scheduling. (2003)
21. Schwarzfischer, T.: Using value dependencies to schedule complex soft-real-time applications with precedence constraints. In: Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications. (2003)
22. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H.Freeman and Company, San Francisco, California (1979)

Appendix A: Proof and Complexity for Single Capacity Problem Algorithm

Proof. There are three cases for minimum duration schedules resulting from greedy approach.

- Case 1. The end time of the last activity has passed deadline, then there is no feasible solution. This is because in this single capacity problem, greedy algorithm achieves minimum make-span schedule. If this schedule can't meet the deadline, no schedules can meet it.
- Case 2. There is no idle period in the schedule, then stop, obviously, the current quality sum is the optimal value.
- Case 3. There is some idle time in the schedule.

We only need to prove the algorithm can find an optimal solution in case 3.

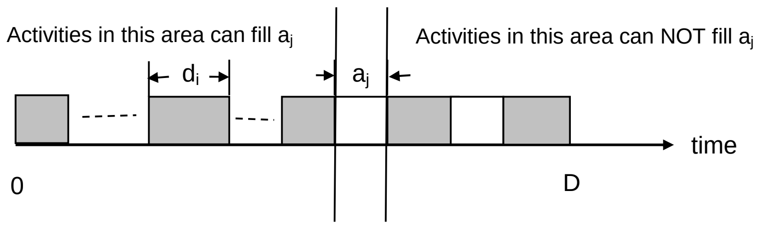
Activities in this area can fill a_j
Activities in this area can NOT fill a_j
d_i
a_j
time
0
D

Fig. 5. Example for Single Capacity Problem

Fig.4 is a schedule in case 3, in which there are several busy periods and idle periods. As before, we denote the minimum duration of activity i as d_i. We denote the length of an idle period j as a_j, and the deadline is D, the time horizon starts at time zero. Then

$$\sum_i d_i + \sum_j a_j = D$$

It is clear that all the schedules assuming activities' minimum durations have the same amount of total idle time $\sum_j a_j = D - \sum_i d_i$, and the same amount of quality$\sum_i k_i * d_i$. Therefore, the key to maximize the final schedule's quality is to allocate the idle time to the most valuable activity which is eligible. This is just what we did in the **while** loop:

Due to the greedy approach, we can't allocate the idle time to the activities after it(as shown in Fig.4), because they are ineligible at that time. So in the **while** loop, we allocate the idle time period one by one from backward. The set of activities before one idle period is a complete eligible candidate set including all the extensible activities for this idle period. The activity chosen with the maximum quality slope is the most valuable activity which is also eligible. So we'll reach the optimal schedule in the end. □

Complexity 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 the increasing order of release dates. If it 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 until 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).

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

Appendix B: Proof of NP-completeness for Quality Maximization Scheduling Problem

Proof. We prove this by reducing a known NP-completeness instance to this problem.

Definition 2. ***Multiple Capacity Single Resource Quality Sum Problem with Linear Quality Profile*** *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 3. ***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[22].

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. We prove finding a feasible solution is NP-complete.

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, $s_i = D - e'_i \geq 0$.