A Precedence-Constraint-Posting Procedure to Generate Schedules Maximizing Quality with Anytime Property

Xiaofang Wang and Stephen F. Smith[2]

CMU-RI-TR-04-25

March 2004

A Precedence-Constraint-Posting Procedure to Generate Schedules Maximizing Quality with Anytime Property

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. Inspired by the popularity of anytime algorithms in solving complex real world problems, we define a class of scheduling problems. In addition to the normal resource constrained project scheduling assumptions, each activity in a project has a quality profile with an anytime property, which means that it can be stopped at any time and the quality of the output is proportional to its duration. Instead of finishing all activities as fast as possible, the goal of scheduling is to maximize the sum of qualities given a hard project due date. We formulate this quality maximization problem as a constraint-based optimization problem, analyze its complexity and present a precedence constraint posting procedure for generating a schedule with a good quality sum. Different constraint posting heuristics are evaluated based on the solution they can produce, CPU time and the number of posted constraints.

1 Introduction

In the past few years, constraint satisfaction problem solving(CSP) techniques have been applied to several classes of scheduling problems[1–7]. Most work aims at generating feasible schedules or optimizing the objective of the make-span or weighted sum of tardiness. Another important, but often ignored measure is the quality of the final product from a process. This quality maximization scheduling problem is motivated by the knowledge intensive production process widely existing in news agencies, hospitals, government agencies, research and development departments.

Let us consider the process of creating a knowledge product(a news story, a new product design, or a solution for a series of complex decision problems etc.) as a project, where activities represent steps that must be performed to

achieve completion of the project. Those activities are subject to precedence constraints. And each activity's productivity is measured by an any-time performance profile, which means it can be stopped at any time and the quality of the output is proportional to its duration. In a knowledge intensive production process, most activities have this property, for example, information retrieval, heuristic search, data analysis and some tasks performed by humans. When raw materials(a news event, input data, etc.) go through different activities, quality is increased according to the corresponding activity's performance profile which is an increasing function of time. The final product's quality is determined by the accumulated quality gains from all activities in the network.

Similar with resource constrained project scheduling problems, we require the final schedule should be subject to and satisfy both precedence constraints and resource constraints. Can we solve this new problem by using traditional constraint satisfaction strategy, for example, Precedence-Constraint-Posting Procedure? To achieve high quality, we prefer long duration for each activity, but that may conflict with the available resource and the hard due date. How can we factor the quality and efficiency trade-off consideration into our *PCP* strategy to find a good schedule measured by both quality and computational cost? Those interesting problems will be investigated in this paper.

From real-world point of view, which activities are worth spending more time? Should we allocate most time to the most valuable task? From the results in this paper, we can see this isn't always true. Many knowledge product creating processes lack guidance about how to allocate resources including people and time to maximize quality with anytime property. This may be because there are no consistent definitions about the quality at each knowledge-processing activity or the performance interdependence among the knowledge-processing activities. Traditional scheduling models, no matter project scheduling or job shop scheduling, haven't defined or solved this problem well. Attempting to fill this gap, we formulate and investigate this quality maximization problem, propose a heuristic based approach within a Precedence-Constraint-Posting framework. We hope the results can help knowledge intensive production managers to evaluate its productivity given the current employees and make correct decisions.

In the following section, we review related research. In section 3, we formulate the quality maximization problem, give the optimization procedure for single capacity resource case and the complexity analysis for multiple capacity resource case. In section 4, we present the precedence constraint posting algorithm and heuristics. In section 5 we report the experimental results. Conclusions and discussions are in the section 6.

2 Literature Review

We first review the constraint satisfaction problem solving (CSP) based scheduling research which provides the context and basis of our solution framework. Then we summarize the related operations research literature, especially the continuous time/cost tradeoff problem, whose solution procedure serves as our

basic optimization subroutine. In the end we briefly summarize related deliberation scheduling research where the concept of anytime algorithm originates.

In the CSP literature, there are two main approaches to formulate a scheduling problem as a constraint satisfaction/optimization problem. In start time assignment model[5, 8], decision variables are the start times of various activities. And in precedence constraint posting(PCP) model[9, 1], decision variables are the ordering decisions that need to be made between sets of activities that are competing for the same resources. [10] gives a good comparison of the two formulations. In our paper, we use the second formulation. Recently, generating flexible schedules received more and more attention. [7, 11] use PCP to generate partial order schedules, schedules that retain temporal flexibility. Similar with their paper, the solution schedule in our paper is presented by the temporal constraints defined in the problem and those added to reach resource feasibility. But a further optimization procedure is used to determine the specific start time and end time for each activity. In other words, the resulting schedule in our paper is a resource and temporal feasible fixed time schedule optimizing the total quality outcome.

Related to our problem, resource constrained project scheduling problem(RCPSP), is to decide start times for the activities of a project in order to satisfy temporal and resource constraints. RCPSP has been widely studied in operations research literature(see the survey paper[12]). Finding a feasible schedule for the resource constrained project scheduling problem alone is NP-hard[13, 14]. A variation of RCPSP is to provide the option of speeding up an activity by spending more on it, which is called time-cost tradeoff problem or project crashing. Its goal is to meet the project's due date, while minimizing total crashing costs. The time-cost tradeoff problem was formulated over forty years ago by Kelley and Walker[15]. If the cost function for each activity is linear and continuous, the problem can be solved in polynomial time by linear or network programming[16–18]. But most time-cost tradeoff problems don't consider resource constraint, which means there is no imposed upper limit for the number of concurrent activities. There are some papers from operations research literature trying to bridge the gap between time-cost tradeoff and scheduling under resource constraints. But they are either non-preemptive case with discrete cost function[19, 20], or preemptive case[21, 22]. To our knowledge, there is no work dealing with resource constrained non-preemptive time-cost tradeoff problem with linear and continuous cost function.

We realize the similarity between time-cost tradeoff problem and our quality maximization problem. In both of them, the duration of each activity should be decided and the durations are linearly related to the objective. We know the uncapacitated time-cost tradeoff problem is solvable. This motivates us to exploit the results and methodology from time-cost tradeoff problem in the design of our precedence constraint posting procedure.

The problem of deliberation scheduling is to allocate the computation time to decision-making procedures based on the expected effect of those allocations on the systems' behavior[23]. The decision-making procedures are anytime algo-

rithms that can be interrupted at any point to provide an answer whose quality is an increasing function of computation time[24, 25]. The function is called performance profile. Because the time spent on computation has a cost, there exists a tradeoff between the decision quality and computation time. Recently Schwarzfischer [26] introduced a new idea of combining deliberation anytime scheduling and deadline scheduling. He called it quality/utility scheduling. Schwarzfischer[27]starts with acyclic task networks on single processor and extended his work into the situation with precedence constraints[28]. Similarly in our paper, we assume each activity has an any-time performance profile. But different from the previous deliberation scheduling research, we focus on multiple capacitated resource(multiple processors) and want to look at the relations between levelling resource and the quality from the schedule. Even with the simplified linear, continuous profile assumption, we will show the precedence and resource interdependence among activities can make the quality-time tradeoff problem very hard.

3 The Quality Maximization Scheduling Problem

Given a set of activities $V = \{a_1, \ldots, a_n\}$, a set of precedence constraints, a set of quality profiles and resource with limited capacity, the 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:

- each activity a_i has a release date r_i,
- all the activities have a common deadline D,
- activities are non-preemptive,
- each activity i has an anytime quality profile which is non-decreasing linear and continuous function of its duration with slope k_i,
- each activity i has a minimum duration d_i.[3]
- the schedule must satisfy the precedence constraints, E is the set of edges in the precedence graph, if $(i, j) \in E$, activity j should start after activity i is completed,
- each activity requires one unit of resource[4] and the capacity of the resource is a constant C over the entire horizon,
- s_i: decision variable, the start time of activity i,
- e_i: decision variable, the end time of activity i,

[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] In traditional multiple capacitated scheduling problems, each activity or task can require more than on unit of resource, but this assumption here still makes sense in knowledge intensive settings, where humans or some decision supporting tools are major resources. If we consider the task using more than one unit of resources, more complicated productivity models are needed in stead of a curve, which is out of current interests of this paper.

– t: current time.

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

$$R(S,t) := \sum_{i \in A(S.t)} 1$$

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

$$q(x,y) = \sum_{j=1}^{n} k_j(e_j - s_j) \tag{1}$$

subject to

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

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

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

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

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

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

3.1 Polynomial Algorithm for Single Capacity ($C = 1$) Problem

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]

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

[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

3.2 Complexity Analysis for Multiple Capacity ($C > 1$) Resource Case

Theorem 1. *Multiple Capacity Single Resource Quality Maximization Scheduling Problem with Linear Quality Profile is NP-complete.*

Proof. : refer to appendix. □

Actually, we can see from the mathematical formulation, the resource constraint is the only constraint that is not easily implemented in a linear programming solver. This is because before we determine the schedule, we don't know which of the activities will be concurrent. Thus we are not able to use inequalities to represent this constraint as input of LP solver. The choices of concurrent activities exponentially increase with the increasing number of activities, which makes this problem very hard.

4 The Precedence Constraint Posting Framework

If we take away the resource constraint, we are left with essentially the time-cost tradeoff problem which can be solved by linear programming. Based on this infinite capacity solution, we detect resource "contention peaks"(see definition 1). Posting sequential constraints between competing activities is used to reduce demand and eliminate resource peaks. After posting a new constraint, linear programming solver is called again to modify some activities' durations in order to optimize quality. Those two sub-procedures alternate until a resource feasible solution is reached or failure is reported, which means given current precedence constraints, no temporal feasible solution exits. The performance of this algorithm depends greatly on the heuristic of choosing conflict and posting new constraints. The procedure is described as follows:

4.1 Heuristic-Based Algorithm for Multiple Capacity Problem

Heuristic-Based Algorithm for Multiple Capacity Problem
Input: A set of activities with constraints.
Output: A solution or indicating it is unable to solve(we have to do backtracking)
1. **loop**
2. apply linear programming solver to find infinite capacity optimal solution
3. **if** there is no temporal feasible solution
4. **then return** unable to solve the problem
5. **else begin**
6. detect resource conflicts
7. **if** there is no conflict
8. **then return** solution
9. **else** resource levelling: sequence a pair of activities in some peak
10. **end**
11. **end-loop**

Fig. 2. Heuristic-Based Algorithm for Multiple Capacity Problem

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* $(t_2 - t_1)$*.*

4.2 Resource levelling heuristics

There is a traditional resource levelling method which involves selecting the next contention peak to resolve and determining how to resolve it(i.e.,which precedence constraints to post in the peak). Following this tradition, we propose three simple heuristics:

- *Random-low.* Randomly choose the peak and post constraints between the two activities with lowest quality slopes in this peak. Sequence them by Earliest-Start-Time first rule.
- *Long-low.* Choose the longest peak and post constraints between the two activities with lowest quality slopes in this peak. Sequence them by Earliest-Start-Time first rule.
- *Short-low.* Choose the shortest peak and post constraints between the two activities with lowest quality slopes in this peak. Sequence them by Earliest-Start-Time first rule.

The reason for us to choose peaks according to their length follows from the observation that an activity's length is closely related to the quality contribution from itself and the other activities running in parallel with it.

4.3 One Step Quality Loss Estimation

When we start with an infinite capacity solution, we actually set an upper bound for the final solution. A good resource levelling heuristic transforming an initial infinite capacity solution into a conflict-free solution should achieve minimum quality loss from that upper bound by posting as few as possible constraints. Myopically, we try to post a constraint which can lead to minimum quality loss in one step. To predict the quality loss, the accurate method is to run linear programming solver(LP) on the problem twice, with and without this constraint separately. The one step quality loss is the difference of the two quality values. But it is very costly in computation time, as shown in our following experiments. If we can predict the quality loss and make a choice without running LP on the new problem, it will save a large amount of time and such a heuristic can be more applicable in reality. So we designed the following estimation methods.

Quality Loss Estimation Method 1 - Loss Only Suppose we choose the partially overlapped activity pair $< a, b >$. 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 the 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.

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

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

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

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

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

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

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

As shown in the example of figure 1, 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. Let A_{loss} be the quality loss due to a's duration, B_{Loss} be the quality loss due to b's duration. Therefore,

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

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

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

If just as we assumed, the other activities stay unchanged after re-optimization on the new problem with new precedence constraint, the above quality loss estimation will be the accurate quality loss. But from the experiments, we see in some cases, if we post a new constraint, the other activities will change, too. So we say this method is an estimation.

Quality Loss Estimation Method 2 - Loss and Gain Actually, 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, b's predecessors or successors will grow longer up to Δa, similarly, a's predecessors or successors will grow longer up to Δb. So we should subtract the quality gain from the above quality loss function. The set of activities which are likely to grow longer are called Set_a and Set_b. For example, in figure 1, Set_a is the set of activity A's predecessors, and Set_b is the set of activity B's successors. The modified quality loss estimation function are as follows:

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

We notice the computation of quality gain is over optimistic because the activities growing longer are usually stuck by their other successors or predecessors so that they probably can not grow Δb or Δa that much.

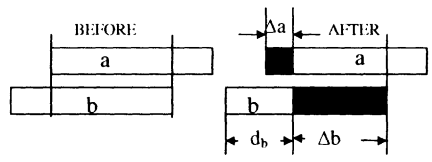

Fig. 3. One Step Quality Loss Estimation

4.4 Heuristic Based on One Step Quality Loss Estimation

- *Step-ratio* First choose globally an activity with largest ratio of duration to slope. Then try to find the other one. Try all possible activities which compete with it for the resource in some peak, predict the quality loss for each of them, choose the one with minimum quality loss, and post the constraint that achieves this minimum quality loss.

The reason of choosing the activity with the largest ratio of duration to slope is to balance the two factors influencing the quality loss: quality slope and reducibility. Reducibility means how much the chosen activity's duration can be

decreased when it conflicts with the other activities. Obviously, an activity with minimum duration doesn't have any reducibility. From this choice, we actually bias on choosing the activity with small slope and long duration.

5 Experimental Evaluation

The test data are based on benchmark single mode resource constraint project scheduling problem set with precedence constraint.[6] There are 498 problems in the file $j30.sm$ and each has 32 activities. We use the precedence constraints defined by this file. 32 uniformly distributed random integers in the range $[1, 50]$ are generated to represent the slopes for each problem. By running experiments, we find the first 200 problems are more resource constrained than the second 200 problems. So we test them separately in order to compare the heuristics' performance for the problems at different levels of hardness. We call the first 200 problems as the hard problem set and the second 200 problems as the easy problem set. The algorithms are implemented in Visual C++ 6.0 and the CPU time presented in the following tables are obtained on a Pentium IV-2.40 Ghz processor under Window XP. We set capacity = 5, due date = 20, minimum duration = 1. Uniformly distributed random integers in the range $[0, 5]$ are generated to represent the release date for each problem[7]. In the following tables, we compare the performance of heuristics based on following measures:

1. *quality(%).* The average percentage quality 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 average percentage number of problems solved, which means the heuristic can guide the search to a resource feasible solution without backtracking.

In table 1 and table 2, we compare the three simple peak selection heuristics, and four step-ratio heuristics which are different in the quality loss computation methods.

- *Step-ratio-accurate* uses the accurate quality loss computation by running the LP solver.
- *Step-ratio-loss* uses the quality loss estimation method 1 without the estimation of quality gain.
- *Step-ratio-loss-gain* uses the quality loss estimation method 2 with the overestimated quality gain.

[6] http://www.bwl.uni-kiel.de/Prod/psplib/data.html.

[7] We choose this range for release dates in order not to make the temporal constraints so tight and guarantee the feasible solution exists for all the problems.

Table 1. Results for the easy problem set assuming capacity = 5

	Quality(%)	Runtime(sec.)	Constraints	solved(%)
Long-low	86.6099	1.01	6.75	99
Random-low	85.7308	1.14	7.58	99.5
Short-low	85.600	0.96	6.40	99
Step-ratio-accurate	85.5323	37.01	15.85	100
Step-ratio-loss	85.4916	1.75	13.18	100
Step-ratio-loss-gain	83.4344	1.11	7.44	100
Step-ratio-gain	82.4307	1.03	6.89	100

Table 2. Results for the hard problem set assuming capacity = 5

	Quality(%)	Runtime(sec.)	Constraints	solved(%)
Long-low	76.6273	1.69	11.86	98.5
Short-low	74.478	2.08	14.61	98.5
Random-low	75.3649	1.44	10.06	98.5
Step-ratio-accurate	73.2177	83.15	30.88	100
Step-ratio-loss	72.3715	3.77	27.26	100
Step-ratio-loss-gain	72.0478	2.01	13.84	100
Step-ratio-gain	70.5268	1.80	12.60	100

– *Step-ratio-gain* choose the second activity maximizing the following measure, which is actually the quality gain in the modified quality loss estimation,

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

From them, we can make several observations:

Observation 1 *With respect to the percentage number of problems solved, we can see the slight difference among the heuristics. All the "Step-ratio" heuristics achieve* 100%*, a little better than "Peak-selection" heuristics on the hard problem set.*

Does this happen by chance or the step-ratio heuristics have the advantage to avoid infeasibility? To see the real performance in terms of the percentage number of problems solved, we need to test the heuristics on harder problems. In the future experiments, we decease the resource capacity from 5 to 3 to make our problems more-severely constrained.

Observation 2 *With respect to the percentage quality, "Long-low" heuristic outperforms all other heuristics in both problem sets. "Random-low" and "Short-low" are slightly worse. Compared with the first three peak selection heuristics, "Step-ratio"heuristics are relatively bad.*

Observation 3 *"Step-ratio-accurate" achieves the best percentage quality among the "Step-ratio"heuristics, but it takes a huge amount of running time compared with others, which is not good for real applications. Compared to "Step-ratio-loss", "Step-ratio-loss-gain" can achieve the comparable quality by posting much less constraints and in a shorter amount of time. Compared to "Step-ratio-gain", "Step-ratio-loss-gain" can post the comparably small number of constraints to reach a better percentage quality.*

Unexpectedly, "Step-ratio-accurate" doesn't have any advantage over the simple "Peak-selection" heuristics, which indicates completely myopic decision may not be a good choice. And because of the huge computational cost, "Step-ratio-accurate" will be eliminated in the following experiments.

Among the "Step-ratio" heuristics, we want to choose the best quality loss estimation method considering all the measures together: number of problems solved, percentage quality, number of posted constraints, and running time. Based on the above observations, "Step-ratio-loss-gain" is the winner . Actually, "Step-ratio-loss-gain" isn't a completely myopic decision. It overestimates the quality gain, because in many cases, the activities in Set_a can't grow as much as Δa, and the activities in Set_b can't grow as much as Δb, they will be stuck by their other successors or predecessors. But the overestimation of quality gain turns out to be a good favor. It actually biases on the more connected activity(the activity with more successors or predecessors). When the search is completely guided by the connectivity(quality gain) just as the "Step-ratio-gain" heuristic does, the solution isn't improved further. So we conclude that combining the information from both the activities which will be posted a constraint between and their connectivity with other activities will achieve a good tradeoff among all the performance measures.

Altogether, we set resource capacity as 3 and will compare the performances of "Step-ratio-loss-gain" with peak-selection heuristics in the following experiments.

Noticing that the above experiments are conducted without back tracking, next we will test the performance when the basic search procedure is integrated into an iterative sampling framework.

- *Long(10 iter.)* choose a peak with the probability in proportional to its length, post a constraint just as "Long-low" heuristic do until a feasible solution is reached, which is called one iteration. 10 iterations are performed, and keep the current best solution.
- *Ratio(10 iter.)* choose the first activity with the probability in proportional to its ratio, post a constraint just as "Step-ratio-loss-gain" do until a feasible solution is reached. 10 iterations are performed, and keep the current best solution.

Table 3 and table 4 show the results for the hard and easy problem sets assuming capacity is 3, and quality percentage is based on the commonly solved problems among the 200 problems. We can make the following several observations:

Table 3. Results for easy problem set assuming capacity = 3

	Quality(%)	Runtime(sec.)	Constraints	solved(%)
Long-low	56.6671	2.72	19.87	58
Random-low	55.2	2.96	21.40	40.5
Short-low	55.6382	2.25	16.43	49.5
Step-ratio-loss-gain	55.2353	3.49	26.06	100
Long(10 iter.)	57.5923	27.85	20.91	81.5
Ratio(10 iter.)	57.3044	32.11	24.17	100

Table 4. Results for hard problem set assuming capacity = 3

	Quality(%)	Runtime(sec.)	Constraints	solved(%)
Long-low	50.725	3.26	24.41	40
Random-low	48.962	3.41	25.63	22
Short-low	49.4034	2.30	17.30	36
Step-ratio-loss-gain	48.9156	4.19	30.78	100
Long(10 iter.)	51.5624	33.67	24.26	67.5
Ratio(10 iter.)	51.1778	39.30	28.85	100

Observation 4 *With respect to the percentage number of problems solved, there is a big difference among the heuristics. The "Step-ratio" heuristic can solve* 100%, *while the fist three simple heuristics solve much fewer problems. When problems become harder, the fist three simple heuristics perform worse, while the "Step-ratio" heuristic is still good. Integrating the basic search in to iterative sampling can improve the number of problems solved for the peak-selection heuristic significantly.*

Observation 5 *With respect to the percentage quality, the first three heuristics are slightly better. Integrating the basic search into iterative sampling can improve the quality of a final solution, but not as obvious as the improvement on the number of problems solved.*

6 Conclusion and Extensions

In this paper, we have defined a quality maximization scheduling problem and investigated the use of precedence constraint posting strategy to schedule activities with anytime property. The problem is proved to be NP-complete. The solution procedure combines constraint-guided heuristic search and linear programming optimization. The constraint posting CSP model iteratively transforms an infinite capacity optimal solution into a resource feasible solution.

Several heuristics have been proposed and tested. The simple baseline "peak selection" heuristics 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 step-ratio heuristics show their big advantage in solving 100% problems no matter what level of hardness the problem is on. This is a promising result in real-world scheduling because if we can easily get a feasible schedule, many local search techniques can be applied to improve this initial schedule and end up with a better one, which is confirmed in our experiments.

Another interesting result is when we try to make a better one step look ahead quality loss prediction, we find the prediction biased on the connectivity of an activity can actually achieve comparable percentage of quality while posting much less constraints and saving large amounts of time. So when there are some resource conflicts in current schedule, it is better for us to sequence those more connected activities, such that the new sequential relation will have more impact on other activities, which reduces the effort to reach the resource feasibility.

From the above result, we also find some managerial insights. Intuitively, a project manager prefer to allocate most time to the most valuable activity. But this can be wrong in many cases. In the presence of concurrency of activities, the decision of an activity's duration should not only be based its own value. We should look at the activities running in parallel with it and its successors or predecessors. So the most valuable activities may not last long and it may leave space to a set of less valuable activities.

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

- *Multiple projects.* In this paper, we look at scheduling within one project, say, creating one news story. But in a news company, there are many story creating projects going on. Scheduling all the activities in different projects will be a more realistic problem and the framework proposed in this work can be extended to solve that problem.
- *More complex quality dependencies.* In our current model, the only dependencies among activities are precedence relationships. Quality dependencies, which mean one activity's output may influence it's successors' quality profiles, are more realistic. In that case, the objective won't be in summation form.
- *Activity Profiles.* Linear profiles can be extended to non-linear concave profiles. Some activities may have fixed durations.
- *Further optimization.* In this paper, we find heuristics which can find a resource feasible solution easily. Based on that, we will use random-restarting framework and local search procedure to further improve the solution.

References

1. Cheng, C., Smith, S.: Generating feasible schedules under complex metric constraints. In: Proceedings 12th National Conference on Artificial Intelligence. (1994)
2. Cheng, C., Smith, S.: A constraint satisfaction approach to makespan scheduling. In: Proceedings 4th International Conference on Artificial Intelligence Planning Systems. (1996)

3. Nuijten, W., Aarts, E.: Constraint satisfaction for multiple capacitated job-shop scheduling. In: Proceedings of the 11th European Conference on Artificial Intelligence. (1994)
4. 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)
5. Sadeh, N.: Look-Ahead techniques for Micro-Opportunistic Job Shop Scheduling. PhD thesis, Dept. of Computer Science, Carnegie Mellon University (1991)
6. Oddi, A., Smith, S.: Stochastic procedures for generating feasible schedules. In: Proceedings 14th National Conference on Artificial Intelligence. (1997)
7. Policella, N., Smith, S.F., Cesta, A., Oddi, A.: Steps toward computing flexible schedules. In: Proceedings $CP-03$ Workshop on Online Constraint Solving: Handling Change and Uncertainty, Cork, Ireland (2003)
8. Nuijten, W., Pape, C.: Constraint-based job shop scheduling with ilog-scheduler. Journal of Heuristics **3** (1998) 271–286
9. Smith, S., Cheng, C.: Slack-based heuristics for constraint satisfaction scheduling. In: Proc.11th National Conference on Artificial Intelligence, Wash DC, AAAI Press (1993) 139–144
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. 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)
12. 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
13. Bartusch, B., Mohring, R., Radermacher, F.: Scheduling project networks with resource constraints and time windows. Annals of Operations Research **16** (1988) 201–240
14. Blazewicz, J., Lenstra, J., Rinnooy Kan, A.: Scheduling subject to resource constraints: Classification and complexity. Discrete Applied Mathematic **5** (1983) 11–24
15. Kelley Jr., J., Walker, M.: Critical Path Planning and Scheduling: An introduction. Mauchly Associates Inc., Ambler,Pa (1959)
16. Fulkerson, D.: A network flow computation for project cost curves. Management Science **7** (1961) 167–178
17. Kelley Jr., J.: Critical path planning and scheduling: Mathematical basis. Operations Research **9** (1961) 296–320
18. Philips, S., Dessouky, M.: Solving the project time-cost tradeoff problem using the minimal cut concept. Management Science **24** (1977) 393–400
19. Talbot, F.: Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. Management Science **28** (1982) 1197–1210
20. 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
21. Slowinski, R.: Two approaches to problems of resource allocation among project activities-a comarative study. J.Operational Research Society **31** (1980) 711–723
22. Slowinski, R.: Multiobjective network scheduling with efficient use of renewable and non-renewable resources. European J.Operational Research **7** (1981) 711–723
23. Dean, T., Boddy, M.: Deliberation scheduling for problem solving in time-constrained environment. Artificial Intelligence **67** (1994) 245–285

24. Horvitz, E.: Reasoning about beliefs and actions under computational resource constraints. In: Third Workshop on Uncertainty in Artificial Intelligence, Seattle, Washington (1987)
25. Dean, T.L., Boddy, M.: An analysis of time-dependent planning. In: Proceedings of 7th National Conference on Artificial Intelligence, St. Paul, MN, AAAI Press (1988) 49–54
26. 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)
27. 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)
28. Schwarzfischer, T.: Application of simulated annealing to anytime scheduling problems with additional timing constraints. In: Proceedings of the Fifth Metaheuristics International Conference. (2003)
29. 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: 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.

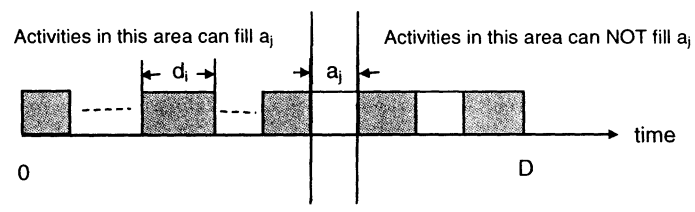

Fig. 4. Example for Single Capacity Problem

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: 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[29].

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

□