Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

A Dynamic Scheduling Algorithm for Time- and Resource-Constrained Task Networks

Qi Hao, Yunjiao Xue, Shuying Wang and Weiming Shen
Centre for Computer-assisted Construction Technologies
National Research Council Canada
London, Ontario, Canada
[qi.hao; yunjiao.xue; shuying.wang; weiming.shen]@nrc.gc.ca

Abstract—**The resource-constrained project scheduling problem (RCPSP) is an extensively explored area. The existing RCPSP approaches tend to focus on single project scheduling problems without practical support to address the multiple project schedule coordination which involves constraints defined across projects. This paper extends RCPSP by involving time and resource constraints and proposes a practical dynamic task network scheduling algorithm. This algorithm takes time constraints, resource constraints, and particularly the dynamic task execution status into consideration. Dynamic scheduling through a partial task network is considered a unique feature of this algorithm. The proposed algorithm is fully implemented and tested in a web-based aircraft inspection maintenance management system.**

I. INTRODUCTION

Scheduling, as loosely defined by Sule [1], involves defining priorities or arranging activities to meet certain requirements, constraints, or objectives. Project scheduling is less difficult if only precedence relationships constrain the activity schedule [2]. However, in practice, activities do not get completed on their own. Instead, they consume resources in the process. Allocating scarce resources among competing activities adds significant complexity to scheduling, which is known as the resource-constrained project scheduling problem (RCPSP) and is NP-hard in the strong sense [3].

Time constraints attached to tasks intentionally leave further time lags between sequential tasks. For example, if a task is defined to "start no earlier than" a certain time, the time then becomes a hard constraint (front point barrier) that the task cannot surpass. The time constraints as well as the resource constraints extend the RCPSP to the problem of time- and resource-constrained project scheduling (TRCPSP). TRCPSP is "over-determined" or "over-constrained" when facing conflicting time and resource constraints.

One example of TRCPSP is aircraft inspection and maintenance. A regular schedule of maintenance services on an aircraft is very important to ensure that the fleet can serve its missions promptly, properly, and reliably within its designed life cycle. In order to maintain an aircraft in a state of "airworthiness", regulations require various kinds of periodical inspections [4]. In the periodical inspections and maintenance practice, major "scheduled" inspection/repair tasks require extensive expertise, constant re-assessment of changing priorities, and frequent re-scheduling in response to changes in personnel, skill sets, equipment availability, and most specifically, to coordinate with the receiving schedule of parts-in-order from the supply department and the repaired components from the external machining shops.

Another big challenge faced by aircraft maintenance facilities is to effectively resolve conflicts caused by shared resources among parallel maintenance projects or crews. The multi-project schedule coordination issue has been specifically addressed in a separate paper [5]. This paper will broadly review the literature in RCPSP researches and applications. A formal description of the TRCPSP is given in Section 3 following a problem specification originated from the aircraft maintenance scheduling problem. We propose a practical dynamic scheduling algorithm based on task network reasoning and heuristics (in Section 4) which takes time constraints, resource constraints and task execution status into consideration. The proposed approach has been fully implemented and tested in a web-based aircraft periodical inspection and maintenance system. Implementation issues are briefly presented in Section 5.

II. LITERATURE REVIEW

Project planning and scheduling has attracted ever growing attention both from science and practice because of the broad applications of the term "*Project*". It is concerned with single-item or make-to-order production where scarce resources have to be met when scheduling dependent activities over time [6].

The RCPSP as a research domain has attracted intense activity for several decades in mathematical modeling (including integer programming, dynamic programming, and branch-and-bound approaches), operations research, constraint satisfaction, heuristics and meta-heuristics based computation methods, for example, genetic algorithm [7], simulated annealing [8], ant colony [9], and swarm theories [10].

Özdamar and Ulusoy [2] have summarized RCPSP approaches according to their objectives, resource types, constraint specifications, time-resource functions, single vs. multiple projects, and optimization vs. heuristics. Brucher et al. [6] conducted a comprehensive literature review on RCPSP approaches and they believed that project scheduling was not comparable to its counterpart of machine scheduling in a common notation and classification scheme. Based on the review and summary of 200 papers, they proposed a new classification scheme to close the gap. Each classified area of project scheduling under resource constraints has been extensively surveyed since the 1960s. Herroelen et al. [11] has directed readers to these early reviews in its references while the survey itself focused on recent progress made with optimal branch-and-bound procedures and their important extensions. Özdamar and Ulusoy [2] looked after the heuristic approaches that they thought were more practical and feasible in solving real-world problems.

To the view of the problem size, only small-sized problem instances with up to 60 activities can be solved exactly in a satisfactory manner, at least for the KSD set [12]. Therefore, heuristic solution procedures remain the only feasible method of handling practical resource-constrained project scheduling problems. Recent overviews of heuristic procedures for the RCPSP can be found in Kolisch and Hartmann [13,14].

III. Problem Description

A. *Problem Specification*

The real world time- and resource-constrained project scheduling problem for aircraft inspections and maintenance is very complex in that:

- A regular project contains thousands of tasks depending on the size of the projects.

- Task precedent relationships define the sequence of tasks. In some cases, a task is required to "pin" on a fixed time on the calendar which is usually called "time constraint".

- Different tasks in parallel ongoing projects often claim the use of shared and limited resources, which are defined as "resource constraints". The following are some examples of resource constraints:
    - Shared equipments and tools
    - Shared staff with different qualifications
    - Shared working spaces with limited access capacity, for example, a cockpit

- Customizable exclusive constraints are implicitly required by practice. Some tasks in aircraft inspection are "exclusive" in nature so that they cannot proceed at the same time. For example:

    "*when draining fuel from an aircraft, all other scheduled tasks for the same aircraft which require electricity must be suspended until the draining is finished.*"

- Several aircraft are served in the same maintenance facility. In other words, schedule coordination of multiple projects is required. For example, the impact of the above draining fuel task can be extended to all tasks across the whole facility, no matter they belong to the same project or not.

- Dynamic re-scheduling needs to be carried out frequently in order to bring project schedules up to date corresponding to changing situations, for example:
    - Tasks that are delayed
    - Tasks that are finished ahead of schedule
    - Shortage of staff in specialty trades
    - Shortage of equipment or tools
    - Delayed arrival of replacement or repaired parts
    - In emergent situations, "exchange of parts" may happen. For example, an aircraft (for example, Project A) can borrow a part already installed on another aircraft (for example, Project B) in order to expedite the progress of Project A.

Above specifications for aircraft inspection and maintenance require a special algorithm that can generate new coordinated project plans considering all within- or cross-project constraints and changing conditions. Due to the harsh requirements on project size, time constraints, resource constraints, and unknown numbers of exclusive constraints, the heuristic approach remains the only feasible method of handling this real-world TRCPSP problem. All approaches in literature have been tested at a small set of activities and constraints, which is not comparable to the scale of the TRCPSP problem specified in this paper. The scope of this paper is dynamic scheduling and allocating conflict resources within ONE aircraft inspection project.

B. *Problem Formalization*

We base our work on the concept of task network. A traditional task network $N = (T, P)$ is composed of a set T of tasks and a set P of precedence relationships between tasks. $T = \{t_1, t_2, ..., t_n\}$, where n is the number of tasks and each t_i is a task, $1 \leq i \leq n$. Among them, t_1 and t_n stand for two dummy tasks which are the beginning and the finish of the project. A dummy task requires no actual work and it helps construct the project hierarchy. The definition of a task network must be COMPLETE. A complete network has no loops, no redundant precedence relationships, and no isolated tasks (a task that has neither preceding tasks nor subsequent tasks).

A task $t \in T$ can be represented as 6-tuple (*ID, D, EST, LST, EFT, LFT*) where *ID* - an unique identifier; *D* - duration; *EST* - earliest start time; *LST* - latest start time; *EFT* - earliest finish time; and *LFT* - latest finish time of the task. We use *t.D*, *t.EST*, *t.LST*, *t.EFT* and *t.LFT* to refer to individual elements. Each task is executed without interruption, which means that the duration of a task cannot be split or leveled. In a traditional task network, *EST* and *LST* are generally used to determine whether a task is on the critical path. For a task $t \in T$, if $t.EST = t.LST$, we say the task is "critical" because its delay will cause the delay of the entire project. The paths composed of critical tasks are called "critical paths". People may also be concerned with *EFT* and *LFT* in a task network; however, in most cases, they can be ignored.

The following paragraphs define the constraints in task networks with further details.

- **Precedence constraints - P**

For $pt, st \in T$, $(pt, st) \in P$ means that pt is a preceding task of st and st is a subsequent task of pt, i.e., st cannot start if pt is not finished. Given a task network $N = (T, P)$ and a task $t \in T$, we use $Pre(t)$ and $Sub(t)$ to denote all direct preceding tasks and direct subsequent tasks of t in N. Formally, we have

$$pt \in Pre(t) \Rightarrow \exists pt \in T: \exists (pt, t) \in P, \text{ and}$$
$$st \in Sub(t) \Rightarrow \exists st \in T: \exists (t, st) \in P.$$

There is no time lag between the tasks in a traditional task network. That is, for any task

$$t \in T,\ t.EST = \max(pt.EFT),\ pt \in Pre(t) \quad (1)$$

Equation (1) expresses the precedence constraints on task t which require that t can only start when all the predecessor tasks $pt \in Pre(t)$ are completed.

- **Time constraints - TC**

There are four types of time constraints that can be attached to a task t. They are:

- START-NO-EARLIER-THAN: $(t, tsne) \in TC, t \in T$
- START-NO-LATER-THAN: $(t, tsnl) \in TC, t \in T$
- FINISH-NO-EARLIER-THAN: $(t, tfne) \in TC, t \in T$
- FINISH-NO-LATER-THAN: $(t, tfnl) \in TC, t \in T$

The time constraints require that the following conditions must be satisfied:

$$\begin{aligned} &t.EST \ge t.tsne\,;\ t.LST \le t.tsnl\,; \\ &t.EFT \ge t.tfne\,;\ t.LFT \le t.tfnl \end{aligned} \quad (2)$$

Considering the precedence relationship $(t, st) \in P$, the four time constraints attached to t in (2) can be reduced to two time constraints attached to t and st as in (3). Fig. 1 illustrates the formulation procedure of the time constraints.

$$\begin{aligned} &t.EST \ge t.tsne\,;\ t.LST \le t.tsnl\,; \\ &st.EST \ge t.tfne\,;\ st.LST \le t.tfnl \end{aligned} \quad (3)$$

- **Resource constraints - RC**

The resources constraints described in III.A are to be extracted to "renewable resources". Suppose that there are K types of renewable resource and the resource constraints applied to a task t can be represented as:

$(t, r_k, \delta_{tk}) \in RC, 1 \le k \le K$, where r_k is a type of resource and δ_k is the capacity that is required by task t on r_k. A type of resource r_k is available with a limited capacity a_k. The constraint that r_k puts on the whole project is that at any time, the number of r_k required by all tasks should be within its maximum capacity/availability a_k.

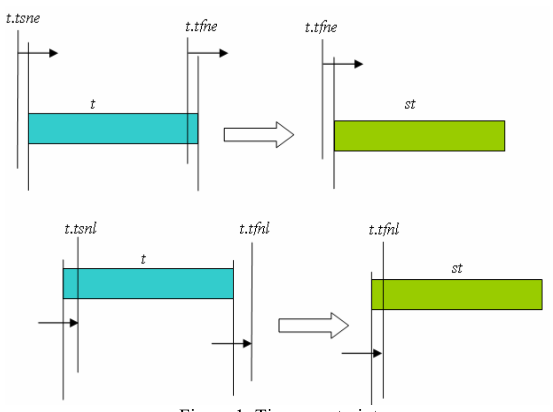

Figure 1: Time constraints

$$\sum_{t_i \in T} \delta_{t_i k} \le a_k, i = 1,2,...,n; k = 1,2,...,K \quad (4)$$

- **Exclusive Constraint**

Suppose that there are K types of exclusive requirements, an exclusive constraint applied to a task t can be represented as $(t, e_k) \in EC, 1 \le k \le K$, where e_k is an exclusive constraint. It must be satisfied that for the whole project, if a task t has an exclusive constraint e_k defined on it, then for $\forall t' \in T$ and $t' \ne t$:

$$\begin{aligned} &((t'.EFT \le t.EST) \cup (t'EST \ge t.EFT)) \cap \\ &((t'.LFT \le t.LST) \cup (t'.LST) \ge t.LFT) \end{aligned} \quad (5)$$

Ideally, the objective of a TRCPSP problem is to minimize the project duration (the finishing time of the unique dummy task t_n), subject to all constraints listed above. However, in practical situations, the objective is to find a feasible rather than optimal solution within a reasonable computation time frame. We have developed a heuristic-based dynamic task network scheduling algorithm to achieve this goal.

IV. A Dynamic Scheduling Algothm for Task Networks

The planning of a project estimates the start time and finish time of each task before the tasks are actually executed. However, projects often do not progress as scheduled. Firstly, task duration is just an estimate so a task may finish earlier or later than expected. Secondly, tasks may be dynamically inserted into, or removed from the project, resulting in changes to the schedule. Thirdly, conflicting task arrangements make the schedule obsolete any time they arise. Therefore, the plan should be dynamically adjusted during the execution of the project.

A. Planning of a Traditional Task Network

Before a network (which represents a project) is planned for the first time, the network will be initialized by creating tasks, creating precedence relationships, and setting all corresponding *EST*s, *LSR*s, *EFT*s, and *LFT*s to *null*.

The algorithm for calculating the *EST* and *LST* of each task uses a two-round reasoning process. In the first round, the algorithm starts from the very beginning of the network (t_1: START_OF_THE_PROJECT), setting both the *EST* and *LST* of t_1 to the given start time of the project. Then, it traverses the network until it reaches the last task (t_n: FINISH_OF_THE_PROJECT). For each task t and its subsequent task $st \in Sub(t)$, $st.EST$ is calculated in the following way:

$$st.EST = \begin{cases} t.D + t.EST, \text{ if } st.EST = null \\ \max(st.EST, t.D + t.EST), \text{ if } st.EST \ne null \end{cases}$$

In the second round, the algorithm starts from the last task t_n, set $t_n.LST = t_n.EST$; and then, it traverses the entire network until reaching the first task t_1. For each task t and its preceding task $pt \in Pre(t)$, $pt.LST$ is calculated as:

$$pt.LST = \begin{cases} t.LST - pt.D, \text{ if } pt.LST = null \\ \min(pt.LST,\, t.LST - pt.D), \text{ if } pt.LST \neq null \end{cases}$$

The traditional task network planning has several limitations:

- The planning of a traditional task network requires a single entry node in the network, which is always t_1 - START_OF_THE_PROJECT. If the project has already started, i.e., some tasks are "finished" or "ongoing", the traditional task network does not support the re-scheduling of the rest of planned tasks. However, in many cases, a project is required to be re-scheduled at any time when necessary, especially when the project is already in progress.
- It only considers the precedence relationships between tasks. It does not support the facts that tasks compete for shared resources, or they may in fact be exclusive to each other. Resolving resource or exclusive conflicts means sequencing parallel tasks by prioritizing them.
- A traditional task network does not include intended time lags (or gaps) between tasks. It is supposed that a task will start right after all its preceding tasks are finished without any delay. However, in reality, intended gaps may be applied between tasks in terms of time constraints or other reasons.

To address the above limitations of a traditional task network, we propose the concepts of *Extended Task Network*, *Partial Task Network* and a dynamic scheduling algorithm based on the partial task network. In the proposed algorithm, a project can be dynamically re-planned based on a given "plan time" and the planning process will only affect the tasks that are not started yet. The planning does not always start from the very beginning of the project; instead, it can start from multiple entry points of a dynamically constructed partial task network. Also, more constraints other than the precedence relationship are resolved in our algorithm.

B. Extended Task Network

An extended task network $N = (T, P, C)$ is composed of a set T of tasks, a set P of precedence relationships between tasks, and a set C of constraints. In an extended task network, a precedence relationship $p \in P$ is defined as a 3-tuple $p = (pt, st, vd)$, where pt is the preceding task, st is the subsequent task, and vd is a virtual duration (or an intended time gap) on the link between pt and st. Therefore, for a task $st \in T$, $st.EST = \max(pt.EST + pt.D + vd)$, where $pt \in Pre(t)$. A constraint $c \in C$ is either a time constraint, a resource constraint, or an exclusive constraint.

C. Partial Task Network (PTN)

Besides making an initial plan for the entire project, it is required to plan just a portion of the project and make sure that the schedule of the planned portion will fit in the entire schedule. A partial task network $N_P = (T_P, P_P, C_P)$ as a part of a full extended task network $N = (T, P, C)$ is composed of a set T_P of tasks, a set P_P of precedence relationship, and a set C_P of constraints, where $T_P \subseteq T$, $P_P \subseteq P$, and $C_P \subseteq C$. For $pt, st \in T_P$, $(pt, st) \in P_P$ means that pt is a preceding task of st. A partial network satisfies the following closure rule:

$$pt \in T_P \rightarrow \neg\exists st \in T_P\text{: } (pt, st) \in P \wedge (pt, st) \notin P_P \qquad (7)$$

Equation (7) means that if one task is in a partial task network, then all of its direct and indirect subsequent tasks should also be included in the partial task network as well. A partial task network reflects a portion of a full task network and the portion itself must be complete.

The partial network is used to calculate the *EST*s and *LST*s of a set of planned tasks while other preceding tasks outside the partial network are not affected. Given a task t in the full task network, its original *EST* and *LST* are set as $t.EST_O$ and $t.LST_O$. If the new *EST* and *LST* values of a partial network are denoted as $t.EST_P$ and $t.LST_P$, then:

$t.EST = max(t.EST_P,\ max(pt.EST_O + pt.D + vd(pt, t)))$, where pt is a preceding task of t in terms of the full task network (not the partial task network) and $vd(pt, t)$ means the virtual duration on the link between pt and t.

Similarly,

$t.LST = max(t.LST_P,\ max(pt.LST_O + pt.D + vd(pt, t)))$.

D. Dynamic Scheduling Algorithm for the Partial Task Network

The process of creating and planning a partial task network includes five major steps as described below.

STEP 1: Finding the heads of the partial network.

To perform a re-planning, the first step is to find the "*heads*" of the partial network, and build a partial network based on the heads. A head is a task that has no preceding task that should be included in the partial network.

For dynamic planning based on a given plan time ι, there are three ways to deal with the tasks:

(1) Finished or ongoing tasks should be discarded;

(2) The tasks that were planned to start later than ι can be moved forward to start from ι if all its preceding tasks are already finished;

(3) The tasks that were planned to start earlier than ι but actually not started yet should be postponed to start from ι.

Fig. 2 shows a scenario when a project needs to be re-planned at time ι. In this scenario task t_1 is finished, task t_2 has started but not finished (ongoing) and all others tasks (t_3, t_4, t_5 and t_6) are planned (not started). The solid or dashed lines between tasks represent the precedence relationships.

At time ι, an observation from rule (3) is that task t_3 and t_5 should be postponed to time ι since their original *EST*s are earlier than ι; and from rule (2), task t_4 and t_6 should be moved forward (or expedited) since their original *EST*s are later than ι. However, t_5 is not a head since t_5 has a preceding task t_3; t_3 is a head and should be postponed to time ι. t_4 is another head and should be pushed forward to time ι because its direct proceeding task t_1 is not included in the partial network according to rule (1). Since t_6 has an ongoing task t_2 that is expected to be finished later than ι, t_6 actually cannot be pushed forward to ι. However, since t_6 has no preceding tasks

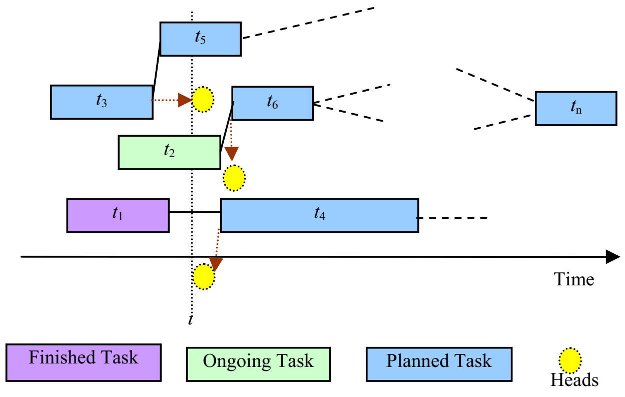

Figure 2: Finding heads for PTN at time ι

that should be included in the partial network, it is also a head. In Fig. 2, the yellow circles indicate that t_3, t_4, and t_6 will be used as heads in the planning process.

The *EST*s of the head tasks need to be determined before the next step. Basically the *EST* of each head task can be set as the given time ι. The exceptional cases are:

(1) A head task to be pushed forward is preceded with ongoing tasks. If the maximum *EFT* of a head task *ht*'s proceeding tasks is denoted as EFT_{maxp} and $EFT_{\text{maxp}} \geq \iota$, *ht.EST* should be set as EFT_{maxp}.

(2) If there is a START-NO-EARLIER-THAN time constraint attached to a head task *ht* and the constrained time is ι_c, then *ht.EST* should be the maximum of EST_{maxp} and ι_c.

For re-planning based on a specific set of given tasks, simply take them as the heads of the partial network. However, only the independent tasks should be taken as heads. All dependent tasks that have direct or indirect proceeding tasks in the given set should be eliminated first.

STEP 2: Creating a partial task network.

After the heads (denoted as a set *H*) are discovered, a virtual starting task *vt* is created and connected to each head task. The duration of the virtual task is set as 0. As the virtual entry node of the network, $vt.EST = vt.LST = \iota$. For any $ht \in H$, a link (*vt*, *ht*) is created and added into P_P. For each link (*vt*, *ht*), a virtual duration is calculated as $vt.EST - ht.EST$.

A complete PTN is then constructed by finding the closure

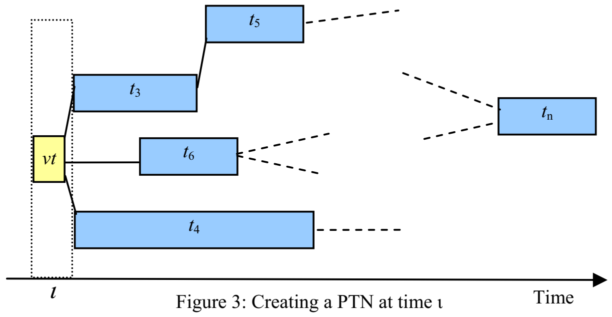

Figure 3: Creating a PTN at time ι

of the head tasks. A closure is composed by all the tasks that have direct or indirect precedence relationships with the head task. The PTN created from Fig. 2 is drawn in Fig. 3. The span of ι is enlarged to clearly illustrate how *vt* is connected to the head tasks.

STEP 3: Calculating the *EST*s of the tasks in the PTN

The algorithm of calculating the *EST*s of all tasks in the PTN will start from the virtual task node *vt*. When calculating the *EST* of a subsequent task *st* of the current task *t*, $st.EST = t.EST + t.D + vd$, where *vd* is the virtual duration on the link between *t* and *st*.

After *st.EST* is calculated, the algorithm validates whether any time constraints attached to *st* is violated. If a task *st* has a START-NO-EARLIER-THAN time constraint defined and the constrained time is ι_c, the algorithm checks the just calculated *st.EST*. If *st.EST* is earlier than ι_c, the task should actually be forced to start at ι_c, i.e., set the virtual duration between *st* and the current task *t* to $\iota_c - st.EST$ and then update *st.EST* to ι_c.

If *st* has START-NO-LATER-THEN time constraint defined and the constrained time is ι_c, but the calculated *st.EST* is later than ι_c, then an irresolvable conflict has occurred. The algorithm has to terminate. Either the time constraints on *st* or the network definition should be modified to avoid such a conflict.

STEP 4. Resolving other conflicts between tasks

For the current task *t*, if *t* has an exclusive constraint defined, or if *t* needs to use a limited resource, then other tasks that are affected by the exclusive constraint or that require the same resource actually cannot be scheduled simultaneously with *t*, even though they do not have precedence relationships. Therefore, when resolving the conflicts, some tasks should be moved to a later time and the virtual durations on links should be updated accordingly.

If task *t* and *t'* overlap in terms of time schedule but *t* and *t'* cannot be executed at the same time, the algorithm applies heuristic rules to determine the priorities of *t* and *t'*. The one with a lower priority should be moved to a later time. Assuming that *t* is to be moved, then *t.EST* is set as *t'.EFT*. Then, for each preceding task *pt* of *t*, the virtual duration *vd* of *pt* and *t* is set as $|pt.EST - t.EST|$. This process is repeated until there is no more conflict remaining.

STEP 5. Calculating *LST*s of tasks

In the final step, the *LST*s of tasks are calculated backwards using the formulation described in the extended task network which considers virtual durations on task links.

V. Implementation and Case Study

Based on the proposed partial task network and dynamic scheduling algorithm, we have developed a web-based system to manage multiple aircraft inspection projects. The system has a three-layer architecture: the presentation layer, which contains a set of JSP pages; the application layer, which is composed of a set of Java classes that implement the system functionalities, execute task network scheduling algorithms, and provide information to the presentation layer; and, the persistence layer, which is built as a MySQL database, providing fundamental data query and persistent storage to the application layer.

The hierarchy of aircraft inspection projects is built in four layers: Project → Phases → Areas → Cards. Again, we involve more pairs of dummy tasks at different levels to help represent the hierarchy. A regular task MUST belong to a "stage" in a project; and it may further belong to a group defined under the stage ("Area" or "Area and Cards").

The system supports creating new projects from pre-defined templates. The inspection tasks included in the templates are usually routine inspection tasks that must be applied to every aircraft under inspection. A sample template contains 700~800 routine inspection tasks and 600~700 dummy tasks for marking the boundaries of 12 stages and roughly 300 groups in the project hierarchy. Users can dynamically add new inspection tasks during any stage of a project.

When creating a new project, the system requires the project manager to provide a planned start time. This given time will be used to create the first plan for the project. Then, the user can request to re-plan a project whenever it is necessary. In practice, re-planning is done repeatedly on a regular basis, for example, the beginning of each day or each week. If a new task is inserted into the project, re-planning will be automatically done by the system to maintain a consistent schedule for all affected tasks. Fig. 4 shows a Gantt chart view of a project schedule.

VI. Conclusion

In project management, more complicated constraints must be addressed when facing real-world problems. Coping with the complexity of TRCPSP is a theoretical challenge although part of this problem (the RCPSP problem) has been an active research topic for several decades. Based on the traditional task network, we proposed the concept of an extended task network and partial network, and a practical dynamic scheduling algorithm for time- and resource-constrained task networks, in which time constraints, resource constraints, exclusive constraints, and particularly the changing project execution conditions are all resolved. In response to emergent events (e.g., absence of maintenance staff and delayed arrival of components), a project schedule can be re-calculated at any time based on the transient partial task networks (PTN) and heuristics. The development of the algorithm uses a bottom-up strategy and a practice-oriented attitude that is very different from those developed from purely academic researches.

The proposed approach is fully implemented and applied to build an aircraft inspection maintenance management system featured with effective decision supports for project dynamic scheduling and conflicts resolution for multiple projects. Currently the developed prototype system is ready for field testing and deployment. Though originally targeting for aircraft inspection and maintenance, our approach can be applied to scheduling problems in a wider area of applications, for example project management and facility maintenance management in industries.

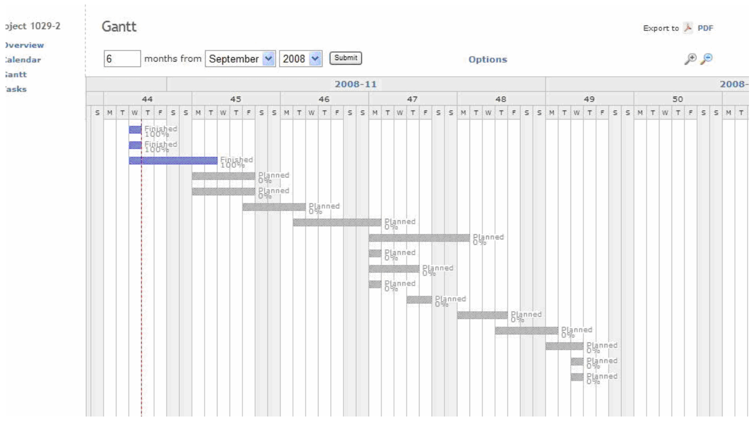

Figure 4: Gantt chart of a small project

References

[1] Sule, D.R. (1997) *Industrial Scheduling*. PWS Publishing Company, Boston, MA.

[2] Özdamar, L. and Ulusoy, G. (1995) A survey on the resource-constrainted project scheduling problem. *IIE Transactions* 27(5):574-586.

[3] Blazewicz, J., Lenstra, J.K., and Rinnooy Kan, A.H.G. (1983) Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics* 5:11-24.

[4] Flight Department Essentials: Helping Your Business Take Flight, Section 11. URL: http://web.nbaa.org/public/ops/adm/fde/#contents

[5] Hao, Q., Wang, S., Xue, Y, Shen, W. (2009) An interactive decision support method for multi-project schedule coordination. *CSCE 2009 Construction Specialty Conference.* May 27-30, St. John's, Newfoundland & Labrador.

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

[7] Vallas, V., Hallestin, F. and Quintanilla, S.A. (2008) A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research* 185(2):495-508.

[8] Safaei, N., Saidi Mehrabad, M. and Jabal-Ameli, M.S. (2008) A hybrid simulated annealing for solving an extended model of dynamic cellular manufacturing system. *European Journal of Operational Research* 185(2):563-592.

[9] Merkle, D., Middendorf, M. and Schmeck, H. (2002) Ant colony optimization for resource-constrained project scheduling. *IEEE Transaction on Evolutionary Computation* 6(4): 333-346.

[10] Zhang, H., Li H. and Tam C.M. (2006) Particle swarm optimization for resource-constrained project scheduling. *International Journal of Project Management* 24(1): 83-92.

[11] Herroelen, W., De Reyck, B. and Demeulemeester, E. (1998) Resource-constrained project scheduling: a survey of recent developments. *Computers and Operations Research* 25(4): 279–302.

[12] Kolisch, R., Sprecher, A., Drexl, A. (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science* 41, 1693-1703.

[13] Hartmann, S. and Kolisch, R. (2000) Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research* 127(2):394-407.

[14] Kolisch, R. and Hartmann, S. (2006) Experimental evaluation of heuristics for the resource-constrained project scheduling: an update. *European Journal of Operational Research* 127(2):394-407.