Task network-based project dynamic scheduling and schedule coordination

Hao, Q.; Shen, W.; Xue, Y.; Wang, S.

Publisher's version / Version de l'éditeur:
https://doi.org/10.1016/j.aei.2010.07.001
Advanced Engineering Informatics, 24, 4, pp. 417-427, 2010-09-01

Task network-based project dynamic scheduling and schedule coordination

NRCC-52705

Hao, Q.; Shen, W.; Xu, Y.; Wang, S.

September 2010

A version of this document is published in / Une version de ce document se trouve dans:
Advanced Engineering Informatics, 24, (4), pp. 417-427, September 01, 2010,
DOI: 10.1016/j.aei.2010.07.001

Task Network-Based Project Dynamic Scheduling and Schedule Coordination

Qi Hao[1], Weiming Shen, Yunjiao Xue, Shuying Wang
Centre for Computer-assisted Construction Technologies
National Research Council, London, Ontario
800 Collip Circle, London, Ontario, Canada; N6G 4X8
Email: qi.hao;weiming.shen;yunjiao.xue@nrc-cnrc.gc.ca; wangvinson@hotmail.com

Abstract: The resource-constrained project scheduling problem (RCPSP) is an extensively explored area. The existing RCPSP solutions tend to focus on single project scheduling problems without practical supports to address complex constraints, dynamic environments, and multi-project schedule coordination. This paper proposes a dynamic project scheduling algorithm based on partial task network heuristics. This algorithm takes time constraints, resource constraints, and particularly the changing task execution status into consideration. To coordinate conflicting schedules of multiple projects, we proposed an interactive decision support process and developed new algorithms for conflict detection, conflict resolution, and impact analysis. The proposed algorithms are fully implemented and tested in a web-based aircraft maintenance management system and are being applied in construction for project schedulingand facilities maintenance management.

Keywords: Resource-constrained project scheduling problem (RCPSP), dynamic scheduling, multi-project scheduling, conflict resolution, aircraft maintenance management.

[1] Corresponding author TEL 1-519-430-7165 FAX 1-519-430-7064

1. 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. 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 [2].

Most research efforts in project scheduling tend to focus primarily on single project scheduling. However, it is essential to extensively address the multi-project scheduling problems because most real life projects involve global resource constraints and the launching of concurrent projects in order to effectively utilize limited resources. The Resource-Constrained Multi-Project Scheduling Problem (RCMPSP) is an extension of the well-known RCPSP problem and it involves the precedence constrained scheduling of two or more projects' activities competing for the same set of scarce resources [3]. RCMPSP is an over-determined problem involving conflicting constraints defined in multiple projects and it is a problem that none of the existing single-project RCPSP approaches can deal with easily.

The RCMPSP problem described in this paper is based on the requirements arising from aircraft inspection and maintenance practices. 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 the periodical inspections and maintenance practice, major "scheduled" inspection/repair tasks require extensive expertise, constant re-assessment of changing priorities, and frequent schedule updates in response to changes in personnel, skill sets, and equipment availability. Aircraft maintenance facilities find it especially challenging to coordinate the schedule of an inspection project with external activities (for example, the receiving schedule of parts-in-order from the supply department) and to resolve conflicts between multiple aircraft inspection projects or crews. Manual coordination of these schedules is

very tedious and every aspect can never be taken care of. The dynamic coordination process requires a tool that can involve the input of all the participants' and the calculation of all possible effects of any decisions.

The classical RCPSP problem has attracted intense activities for several decades in different academic disciplines and industries. However, all approaches in literature have been tested in a small set of activities and constraints, which is not comparable to the scale of the RCMPSP problem of aircraft inspection projects specified in this paper. A dynamic project scheduling algorithm, based on partial task networks, is proposed to solve large scale RSPSP problems with complex constraints, such as time, resource and exclusive constraints. This algorithm forms a solid basis for RCMPSP by guaranteeing the validity of the schedule within a project and screening out the complexity when facing multiple projects.

The key to schedule coordination of multiple projects is to detect conflicts and resolve the conflicts through a decision-making process. Multi-project schedule coordination can be achieved through an interactive approach by considering a reduced constraint set (global constraints) and shifting the schedule of a smaller set of tasks. In general, the basic decision options are: "prioritizing" (projects or tasks), "crashing" (tasks), "shifting" (tasks), and "releasing" (constraints). Whatever the options and choices would be, a manager feels hard-pressed to take action because one cannot foresee the ripple effect of the choices in the long run. To assist human decision makers in project coordination processes, this paper proposes the use of an interactive decision support method to gain the balancing of multiple objectives – time, resources, and performance for the coordination of multiple concurrent projects.

The remainder of the paper is organized as follows: Section 2 provides a review of related literature in RCPSP and RCMPSP; Section 3 specifies the requirements and constraints arising from aircraft inspection and maintenance practices; the RCMPSP problem is then formalized in Section 4; the concept of a partial task network and a task network-based dynamic project scheduling algorithm is described in Section 5; a novel

interactive approach for multi-project schedule coordination is proposed in Section 6; Section 7 introduces the developed Web-based aircraft maintenance management system; and the final conclusions are provided in Section 8.

2. Research Literature

Plentiful approaches or solutions have been proposed in the literature to solve the RCPSP problem using mathematical modeling, constraint satisfaction, heuristics and meta-heuristics based computation methods. Herroelen et al. [4] directed readers to several early reviews since the 1960s while the survey itself focused on the recent progress made with optimal branch-and-bound procedures and their important extensions. Based on the review and summary of 200 papers, Brucker et al. [5] proposed a classification scheme according to scheduling environments, activity characteristics, and objective functions to achieve a common notation and classification scheme in the project scheduling domain. Recent overviews of heuristic procedures for the RCPSP can be found in Kolisch and Hartmann [6,7]. Their conclusion is that, in general, meta-heuristic methods outperform rule-based heuristic methods.

Only a few researches have focused on RCPSP problems in the scope of multiple projects. Because of the complexity of RCMPSP problems, priority rule-based heuristics become the only feasible method to construct a feasible algorithm. Katsavounis [3] formulated multiple projects as a single-entry, single-exit weighted directed acyclic graph and applied a single-pass, parallel scheduling heuristic on top of the standard critical path calculation of each individual project. This heuristics-based approach was tested on a small test base with three concurrent projects and 9-12 tasks in each project. Khattab and Soyland [8] believed priority-based rules outperform CPM-based rules used in commercial packages (i.e. Primavera^{TM}) in terms of levelling limited resources among multiple construction projects. Meta-heuristic technologies (for example, genetic algorithm models in Liu et al. [9]) are also applied to the multi-project scenario, where multiple projects' schedules are achieved by a combined global objective function above the project level.

A number of researchers have proposed the use of distributed intelligence (specifically, agent technology) to handle the complexity of the RCMPSP problem. A multi-project planning and scheduling system was developed in [10] in which each project in the proposed multi-agent framework is presented physically by a project agent. A negotiation-based planning and control mechanism was developed to coordinate these distributed project agents. However, global shared resources are not addressed; rather, the resources are scheduled and balanced within each project. Brown and McCarragher [11] proposed a negotiation-based distributed resource conflict resolution approach between maintenance agents, process units and other entities in order to coordinate maintenance and production processes in a manufacturing environment. Results have shown reductions in conflict of over 60% compared to a fixed maintenance schedule. DISA (Distributed Interactive Scheduling with Abstractions) [12] employed a dynamic multi-agent architecture to address the uncertainties in real-world domains. Temporal abstractions, in the form of summarizations and generalizations, are applied to agents in different hierarchies for problem reduction and conflict resolution. The system involves the human-in-the-loop through interactive user interfaces and user interactions.

Another community that extensively research into project planning and scheduling is the construction sector. Betts and Lansley [13] reviewed all of the articles published in the Journal of Construction Management and Economics (CME) from 1983 to 1992. They indicated that these articles published by CME are mainly concerned with production-related issues in the construction industry. In the ASCE Journal of Construction Engineering and Management (CEM), the topic of “project planning, scheduling and systems” in the construction domain was comprehensively investigated for the period of 1983-2000 [14,15]. It is identified that the issue of time scheduling is common in construction and “scheduling” is the leading research topic which has received considerable attention internationally with 4.65% of the 879 articles analyzing scheduling-related problems.

More recently, a knowledge map was developed for construction scheduling [16]. Although the target scope is the scheduling solutions in the construction sector, the

classification scheme proposed in the knowledge map is useful for general RCMPSP problems. Sriprasert and Dawood [17] developed a Lean Enterprise Web-based Information System that addressed some special needs for project management in construction: multi-constraint information management, visualization, 4D modeling and simulation, mobile data retrieving and data collection, etc. A recent paper [18] questioned the utility of traditional schedule approaches to adequately capture the contraints and complexities of the contruction process with respect to schedule generation and management over the course of the project. The authors propose a formal schedule mapping approach to tackle the special problem of distributed schedule coordination between general contractors (GC) and subcontractors (Subs) in typical contruction projects. Shen et al. [19] conducted a comprehesive review of integration and collaboration technologies in AEC/FM with BIM and project management being the main integrators for the building's life-cycle processes.

The interactive decision support method proposed in this paper advocates the involvement of humans in the decision-making process. Compared with [12], our solution is based on practical heuristics for conflict detection, project prioritization and conflict resolution; it has no strictly divided hierarchical temporal abstractions and severance of functions between long-term, middle-term and short-term scheduling horizons. Compared with the scope of [18], our approach tries to achieve true multiple project schedules, instead of integrating a number of distributed schedules into a master project schedule by matching the gaps between terminologies, numbers and level of details of activities.

3. Problem Specification

One of the applications of RCMPSP is aircraft inspection and maintenance. In order to maintain an aircraft in a state of "airworthiness", regulations require various kinds of periodical inspections. In periodical inspections, major scheduled inspection tasks need to be performed in a predefined order to an aircraft that is temporarily taken off its missions.

Conflicting schedules caused by shared resources or other constraints across projects are the major issue to be resolved in order for the whole schedule to be practical. The RCMPSP in aircraft inspection and maintenance is very complex in that:

- A regular project contains thousands of tasks depending on the size of the projects and areas of applications.
- Several projects often claim the use of common and limited resources, which are defined as resource constraints. If concurrent tasks (in different projects) that require the same resource are scheduled to roughly the same time frame, the system needs to find a way to serialize them according to their priorities. The following are some examples of resource constraints:
    - Shared equipment and tools
    - Shared staff with different qualifications
    - Shared working spaces with limited access capacity
- There are customizable exclusive constraints. Theoretically, tasks can be executed simultaneously when they do not have precedence relationships and they are not competing for a common resource. However, it is possible that these tasks are still “exclusive” in nature so that they cannot be preceded at the same time. For example, a painting job on an aircraft requires that, within the whole maintenance facility, all other scheduled tasks which require electricity must be suspended until the painting job is finished. In other words, a painting job might have an impact on the time schedules of all ongoing aircraft inspection projects.
- Coordination of multiple projects is required because of global exclusive constraints and shared resource constraints. The focus of this paper is conflict detection and resolution algorithms for multi-project scheduling coordination.
- Dynamic re-scheduling needs to be carried out frequently for projects considering:
    - 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.

4. Definition of the RCMPSP Problem

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. $T = \{t_1, t_2, \ldots, t_n\}$, where *n* is the number of tasks and each t_i is a task, $1 \le i \le n$. Among them, t_1 and t_n stand for two dummy tasks which are the start and the finish of the project. 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 multi-task network is a set of task networks represented by $MTN = \{N_1, N_2, ..., N_M\}$, where *M* is the number of task networks and each network is independent of each other.

A task $t \in T$ contains the following attributes:

- *ID* - an unique identifier
- *D* - duration
- *EST* - earliest start time
- *LST* - latest start time
- *EFT* - earliest finish time
- *LFT* - latest finish time
- Priority - task priority is an integer between 1 and 9.
- Critical - a critical task is a task $t \in T$, if *t.EST = t.LST*, or the sla**c**k between *t.EST* and *t.LST* is zero. The delay of a critical task will cause the delay of the entire project. The paths composed by critical tasks are called “Critical Paths”.
- Split-able - this Boolean value specifies whether the duration of this task is allowed to be split or not. For example, a non-split-able task has to be arranged in a continuous time period (even when its duration spans over silent or overtime hours). A task is “split-able” by default.

A task $t \in T$ in network *N* contains the following constraints:

- Precedence constraints - ***P***

Given a task network *N = (T, P), f*or *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*; or *st* cannot start if *pt* is not finished. Meanwhile, a task cannot start until all its preceding tasks are finished. For 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*)$\in P$, and *st*$\in$*Sub*(*t*)$\Rightarrow\exists$*st*$\in T$: $\exists$(*t*, *st*)$\in P$

- Time constraints - ***TC***

  Two types of time constraints can be attached to a task *t*. They are:

  - *START-NO-EARLIER-THAN:* $(t, tsne) \in TC, t \in T$

  - *START-NO-LATER-THAN:* $(t, tsnl) \in TC, t \in T$

  Thus, the time constraints can be described as: $t.EST \geq t.tsne$; $t.LST \leq t.tsnl$

  Theoretically, there should be "*FINISH-NO-EARLIER-THAN"* and *"FINISH-NO-LATER-THAN"* to constrain the finishing time of task *t*. However, these two time constraints attached to *t*'s finishing time can be easily transferred to constrain *st*'s starting time for $(t, st) \in P$.

- Resource constraints - ***RC***

  Suppose that there are *K* types of renewable resources and the resource constraints applied to a task *t* can be represented as $(t, r_k, \delta_{tk}) \in RC, 1 \leq k \leq K$, where r_k is a type of resource and δ_{tk} is the capacity that is required by task *t* on r_k. The constraint that a resource r_k puts on the whole project is that at any time the requirement of all tasks to the resource should be within its maximum capacity/availability.

$$\sum_{t_i \in T} \delta_{t_i k} \leq a_k, i = 1,2,..., K$$

  The assumption we made is that no preemption is allowed; which means a task occupies the required resources with the required capacity for the entire period of task duration. And also the resources are "global" in that they are shared by all projects.

- Exclusive Constraint - ***EC***

Suppose that there are *L* types of exclusive requirements, an exclusive constraint applied to a task $t \in T$ can be represented as $(t, e_l) \in EC, 1 \le l \le L$, where e_l is an exclusive constraint. It must be satisfied that for the whole project, if a task t has an exclusive constraint e_l, then for $\forall t' \in T$ and $t' \neq t$:

$$(t'.EFT \le t.EST) \cup (t'\ EST \ge t.EFT)) \cap ((t'.LFT \le t.LST) \cup (t'.LST) \ge t.LFT)$$

Depending on the scale of its impact, an exclusive constraint could be "Local" - ***LEC*** or "Global" - ***GEC***. The former refers to a type of exclusive constraint that will only affect the tasks within the same task network; while the impact of the second type of exclusive constraint will expand to affect all tasks within the *MTN*.

A distinguished dynamic scheduling algorithm is developed based on partial task network heuristics to resolve the constraints of ***TC***, ***RC***, and ***LEC*** within the scope of each project. As for the global constraints introduced by ***RC*** and ***GEC***, we proposed an interactive conflict resolution method to detect conflicts, provide alternatives, and calculate and compare the "ripple effects".

5. Partial Task Networks and Project Dynamic Scheduling

Based on a wide array of literature on scheduling, Smith [20] noted that, in spite of great achievements, we still lack the fundamental capabilities to generate sound schedules under huge problem size and complex constraints, and to manage changes in unpredictable and dynamic execution environments. For a schedule to be of real value to industry applications, schedule updates have to keep pace with the changing environment.

A dynamic RCPSP algorithm for project scheduling is needed when facing the aircraft periodic inspection and maintenance problem. Firstly, task duration is just an estimate so a task may finish earlier or later than expected. Secondly, the identified repair tasks during inspection need to be dynamically inserted into, or removed from the project, resulting in changes to the schedule. Lastly, but most importantly, conflicting task arrangements violating ***TC, RC*** or ***LEC*** make the schedule obsolete any time they arise.

Based on the traditional "Critical Path" reasoning, we propose the concept of an extended task network and a partial task network and implement a heuristic 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.

5.1 Partial task network (PTN)

In order to resolve the constraints added to a 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 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 $pt \in \Pr e(t)$. A constraint $c \in C$ is either a time constraint, a resource constraint, or an exclusive constraint.

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 $T_p \in T, P_p \in P, C_p \in C$. For any $pt, st \in T_p$, $(pt, st) \in P_p$ means that *pt* is a preceding task of *st*. A partial network satisfies the following closure rule:

$$pt \in T_p \rightarrow \neg \exists st \in T_p : (pt, st) \in P \cap (pt, st) \notin P_p$$

This equation 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:

$t.EST = max(t.EST_P, max(pt.EST_O + pt.D + vd(pt, t)))$,

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

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*.

5.2 Dynamic Scheduling Algorithm for PTN

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-scheduling, 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 are not actually started yet should be postponed to start from ι.

(Insert Figure 1 somewhere here.)

Figure 1 shows a scenario when a project needs to be re-scheduled at time ι. In this scenario task t_1 is finished, task t_2 has started but not finished (ongoing) and all other 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 that should be included in the partial network, it is also a head. In Figure 1, 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 of the head tasks. A closure is composed of all the tasks that have direct or indirect precedence relationships with the head task. The PTN created from Figure 1 is drawn in Figure 2. The span of ι is enlarged to clearly illustrate how *vt* is connected to the head tasks.

(Insert Figure 2 somewhere here.)

STEP 3: Calculating the ESTs 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. Not all time constraints can be resolved using the algorithm.

- Suppose *st* has a START-NO-EARLIER-THAN time constraint defined and the constrained time is ι_c. The algorithm checks whether the just calculated *st.EST* is earlier than ι_c, if so the task will actually be forced to start at ι_c, i.e., set the virtual duration between *st* and the current task *t* to ι_c – *st.EST* and then update *st.EST* to ι_c.
- Suppose *st* has a 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 unless the hard time constraint is released.

STEP 4. Resolving other conflicts between tasks

For the current task *t*, if *t* has an exclusive constraint defined, or if *t* requires the use of 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, among the conflicting tasks, some need to be shifted to other time slots 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'*. 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 LSTs 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.

6. The Interactive Approach for Multi-project Schedule Coordination

The assumptions for multi-project coordination are: 1) all projects involved have already been re-scheduled starting from the same time point; and 2) all conflicts within a single project have already been resolved using the dynamic task network scheduling algorithm described in Section 5. This means that before entering the multi-project coordination procedure, the schedule of each project is good for execution without considering other projects. Meta-heuristic based algorithms are preferred by many researches to solve the multi-project scheduling problem and we have developed a novel complete local search with memory approach for Facility Maintenance Management recently [21][22]. However, in this paper, we intend to focus on introducing a more practical interactive multi-project schedule coordination approach that was used in aircraft inspection and maintenance. Before introducing our solution for multi-project schedule coordination, some concepts need to be clarified.

6.1 Active task, conflict

A time window *TW* is a time period defined by a lower time bound and an upper time bound, i.e. *TW* = $[LowerT, UpperT]$. Active tasks (*AT*) are a set of tasks that have either start time or end time or both start time and end time falling into or coving the time window.

$AT = \{t_1, t_2, \ldots, t_x\}$ $\exists t_i \in MTN, 1 \le i \le x$ and

$$((LowerT \le t_i.EST) \cap (t_i.EFT \le UpperT)) \cup ((t_i.EST \le LowerT) \cap (t_i.EFT \le UpperT)) \cup$$
$$((t_i.EST \ge LowerT) \cap (t_i.EFT \ge UpperT)) \cup ((LowerT \ge t_i.EST) \cap (t_i.EFT \ge UpperT))$$

A conflict *cf* is defined by the following five attributes:

- ID - an unique identifier
- CauseTask – defined by a task that has a global exclusive constraint or a resource constraint
- AffectedTaskList - a task list which contains all the tasks in other projects that will be affected by the *CauseTask* at the same time period.
- Type – conflicts may be caused by two types of constraints: “exclusive” (*GEC*) or “resource” (shared resource constraints).
- Status - the status of conflict, either “solved” or “unsolved”

Actually, a *cf* stands for a group of conflicts that is caused by a common *CauseTask*. It includes the pair-wised conflicts between the *CauseTask* and each of the affected tasks on the *AffectedTaskList*. Depending on the scale of overlap situations, some minor schedule overlaps can be ignored in order to reduce the number of conflicts. This is set through the overlap tolerance rate *otr* representing the percentage of overlap allowed between conflicting tasks.

6.2 Algorithms for multi-project schedule coordination

Our solution for multi-project coordination can be broken into several tasks: to detect conflicts for the given projects in a given time window; to solve the identified conflicts according to the priority rules or the modified task attributes; to create updated project schedules; and to assist the decision-making process by analysing the impact of

decisions and comparing different trials. In fact, for the same scenario, users can apply different strategies to resolve conflicts and get different resolution results. We call the whole decision-making process a “trial”; and a trial contains all the information concerning active tasks, conflicts, strategies/rules, partial resolution options and results and final solutions.

We define a task that has an exclusive constraint as an “Exclusive Task”. The effect of the exclusive task on other tasks is described as a set of “Affected Tasks” which belong to either the same project or other projects. By default, all tasks are affected by an exclusive task unless explicitly specified.

6.2.1. Conflict Detection

A time window is required to detect schedule conflicts between multiple projects. The system will then try to detect and resolve the conflicts between tasks that fall within this time window. There are two reasons to define a time window.

1) For large scale projects, multiple projects planned in the same time horizon may have tens of thousands of tasks. Finding and resolving the conflicts in this huge search space is uncontrollable. Using a time window to reduce the search scope and reduce the number of conflicts is a practical and efficient approach.
2) In a dynamic scheduling environment, the status of a schedule is constantly changing with the availability of resources and the actual execution of tasks. Re-scheduling a project is a frequent practice that is necessary for a well-managed organization. Similarly, the coordination of multiple projects in their full life span is not useful since everything is changing frequently. The parallel projection for tasks outside the time window is a reasonable simplification of the problem.

The algorithm to detect conflicts for a given time window is described in Figure 3. In this algorithm, the set of active tasks are firstly scanned to find all the tasks that have exclusive constraints (*GEC*) or resource constraints and separate these tasks into two lists: *el* for exclusive tasks and *rl* for resource tasks. For each exclusive task $t \in el$, all tasks $t' \in AT, t' \neq t$ are checked if they are conflicting with *t* during the time period of

[*t.EST, t.EFT*]. Then, the list of affected tasks is calculated (*cf.affectedTaskList*) by selecting those overlapping tasks that have an *OTR* greater than the number specified by *otr*. After that, if the affected task list is not empty, a conflict is found and appended to the conflict list *CL*. Similarly, the resource-type conflicts are calculated and added to the conflict list as well. Finally, all the conflicts in the *CL* are sorted according to the time sequence.

6.2.2. Conflict Resolution

Conflict resolution is done through an iterative decision-making process involving the human-in-the-loop. Theoretically, the target of conflict resolution is to "serialize" the conflicting tasks in time. Conflicts are resolved one after another by applying a set of default or selected priority rules; either allowing the system to solve all the conflicts or involving human interactions in every step. After a group of conflicts caused by the same *CauseTask* is resolved, conflicts are generally shifted so the system needs to detect conflicts again in order to solve remaining conflicts. This is an iterative and interactive process until all conflicts are resolved.

Different priority rules can be used to "serialize" the overlapping tasks in CL. By default, an exclusive task always has a higher priority than other tasks. However, when two exclusive tasks are in conflict, the system has to apply certain rules (default or user-selected) to decide the priorities of the two tasks. For any two exclusive tasks, *t* and *t'* are in conflict, the following rules can help the user decide if t has a higher priority.

- Closest To Project End Date: $(t_n.EFT - t.EFT) < (t_n'.EFT - t'.EFT)$
- Task Priority: *t.priority>t'.priority*
- Earliest Planned Start Time First: *t.EST>t'.EST*

In addition to priority rules, conflicts can be indirectly resolved by changing the attributes of the critical tasks.

- "Crushing" the task duration. Squeezing the duration of critical tasks (for example, an exclusive task) can significantly reduce the number of conflicts in the conflict list and thus reduce its impact on other projects.

- "Shifting" tasks. Changing the earliest start time of a task (*t.EST*) can move a task to another non-conflicting time slot so when the system re-checks the conflicts, the conflict list shifts, too.
- "Releasing" task constraints. Removing an unnecessary *GEC* definition of an exclusive task, or changing the resource requirement of a task to an alternative source, can release constraints.
- "Prioritizing" tasks. For individual tasks, this is done,obviously, through changing *t.priority*.
- Overtime arrangement. By changing a task to a non-split-able task, or opening up an additional work shift, a user can make overtime arrangements for critical tasks so that the impact of these tasks can be minimized.

After selecting a priority rule or changing a task's attributes, the system needs to update the schedules for all relevant projects. Figure 3 shows an example of a schedule updating for three projects during a conflict resolution process. In this example, an exclusive conflict *cf* is defined as <'*cf001', t2, {t5, t6, t7, t9, t10}, 'exclusive', 'unsolved'*>, where: '*cf001*' is the identification; *t2* is an exclusive task; {*t5, t6, t7, t9, t10*} is the list of affected tasks that are in conflict with task *t2*; 'exclusive' means that '*cf001*' is an exclusive conflict; and '*unsolved*' shows the status of '*cf001*'; it is not solved yet. In STEP 1, as both *t2* and *t5* are both exclusive tasks, the system (or the user) has to decide which task has a higher priority using the rules set in 2a. If *t2* has a higher priority, the system will set the ESTs and EFTs of all the affected tasks (including tasks *t5, t7, t9* and *t10*) to time slots after *t2.EFT* and make updates to each affected project schedule accordingly in STEP 2. The status in *cf* will be changed to '*solved*' after this step. After project schedules are updated, conflicts are re-checked by the system and a new exclusive conflict cf' is identified as *<'cf002', t5, {t3, t6, t7, t9, t10}, 'exclusive', 'unsolved'*>. Since an exclusive task always has higher priority than other non-exclusive tasks, this conflict can be solved as shown STEP 3.

It seems all conflicts are solved after STEP 3. However, since all tasks in the affected projects (*t5* and *t7* Project 2; *t9* and *t10* in Project 3) are moved backwards, the impact of a conflict resolution process on the related projects is always "negative" - postponing

the project schedules. This is a situation that a manager never wants to see. In reality, any decision made is a trade-off between positive and negative results.

The final step justifies a valuable solution in the conflict resolution process by bringing in "positive" effects to project schedules. As shown in STEP 4, the system does a final check and moves a short task *t7* forward in Project 2. In detail, when updating Project 2's schedule, a virtual task *vt* is added before *t5* and *t7* in the partial network and *vt.EST = vt.EFT = t4.EFT*. The system finds *t7* could fill the time gap between *vt* and *t2* because *t7.EST >= vt.EFT* and *t7.EFT <= t2.EST* and sets *t7.EST = vt.EST = t4.EFT*.

The ripple effects, in response to a decision, can be calculated by re-planning each of the projects involved (projects that need to update schedules). Partial task networks for the projects involved are dynamically created starting from a set of the identified independent head nodes. The schedule is then recalculated using the dynamic scheduling algorithm described in Section 5.

6.2.3. Impact Analysis

Impact analysis provides a reasonable time projection of all related projects in the time window. The impact on the project schedule is the most valuable data for decision support especially when thousands of tasks from different projects are taken into consideration. Through impact analysis, the user can evaluate whether he/she has made a good decision in the previous step.

The impact on a project can be a positive or negative value calculated by $t_n EFT' - t_n EFT$, where $t_n EFT'$ is the newly calculated project finish time and $t_n EFT$ is the project finish time of the previous step or the original project finish time.

Conflict resolution is an iterative process to detect conflicts, resolve conflicts and update schedules until all conflicts are resolved. The purpose of impact analysis is to provide meaningful support after every decision to solve a conflict or to change task attributes.

7. Implementation and Case Studies

We have implemented the concepts and algorithms proposed in our work to address the RCMPSP problem in the aircraft inspection and maintenance domain. A fully functional web-based system was developed to manage 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 a 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 system supports creating new projects from pre-defined templates. The inspection tasks included in the templates are usually routine tasks that must be applied to every aircraft under inspection. A sample template contains 700 to 800 routine inspection tasks and 600 to 700 dummy tasks for marking the boundaries of 12 major inspection groups ("phases") and roughly 300 lower-level inspection groups (or "cards") 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 (this plan can be used as a baseline for project performance review). 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. Figure 5 shows a Gantt chart view of a project schedule.

The coordination of multiple projects starts with an interface to input a time window and select projects in scope. The system then comes out with a list of conflicts detected (Figure 6). Conflicts are organized into groups where each group has the same *CauseTask* (the highlighted tasks). Each task in the conflict list is provided with a link as a quick reference to modify its attributes. The conflicts can be resolved by the system, automatically or by the user, interactively. Also, the impact can be seen in each step (marked “1”) or the summary of impacts can be viewed at the end (marked “2”).

Figure 7 shows the summary of the conflict resolution results after three trials. An ideal system needs to record all status information along with the decision-making process, including the transient partial networks. However, this requires explosive usage of system memories and unmanageable calculation time. In our algorithms, we provide impact analysis in each trial by recording all related information but the partial networks, including conflict lists, task lists, rules and the resulting impact. In the solution summary page, the user can 1) open up another new trial (“Try Another Solution”) for the same scenario; 2) select and view a solution in detail; 3) delete a solution from the memory. During a conflict resolution period, the user can also switch from one trial to another to compare the results of different solutions and select the best solution as their final decision.

To validate the generality of the proposed approaches in this paper, we have successfully developed a web-based facility maintenance management system and a case study in construction [21]. The development cycle of this FMM system was significantly reduced due to the direct applicability of the algorithms developed in this paper. In order to improve the quality of solutions for multi-project coordination, we have developed a novel complete local search with memory technique using meta-heuristics formulated by an optimization objective function [22].

8. Conclusion

The resource-constrained multi-project scheduling problem (RCMPSP) is a NP-Hard problem in a strong sense. In large manufacturing or construction project scheduling, the objective is to find a feasible, rather than optimal solution, within a reasonable computation time using rule-based heuristics. A dynamic project scheduling algorithm based on partial task networks is proposed that perfectly solves the large scale RSPSP problem with complex constraints such as time, resource and exclusive constraints. This algorithm forms a solid basis for RCMPSP by guaranteeing the validity of the schedule within a project and screening out the complexity when facing multiple projects. Therefore, multi-project schedule coordination can be achieved through an interactive approach by considering a reduced constraint set (GEC and shared resources) and shifting the schedule of a small set of tasks.

Our solution for multi-project coordination can be broken into several tasks: to detect conflicts for the given projects in a given time window; to solve the identified conflicts according to the priority rules or the modified task attributes; to create updated project schedules; and to assist the decision-making process by analysing the impact of decisions and comparing different trials.

The proposed concepts and algorithms have been fully implemented and tested in a web-based aircraft periodical inspection and maintenance system. Now, the proposed concepts and algorithms have been successfully applied in construction for construction projects and facilities maintenance management.

Reference

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

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

[3] S. Katsavounis, Scheduling multiple concurrent projects using shared resources

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

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

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

[7] R. Kolisch, S. Hartmann, Experimental evaluation of heuristics for the resource-constrained project scheduling: an update, European Journal of Operational Research 174 (1) (2006) 23-47.

[8] M.M. Khattab, K. Soyland, Limited resource allocation in construction projects, Computers and Industrial Engineering. 31 (1-2) (1996) 229-32.

[9] T. Liu, M. Liu, Y. Zhang, L. Zhang, Hybrid genetic algorithm based on synthetical level of resource conflict for complex construction project scheduling problem, Proceedings of the IEEE International Conference on Machine Learning and Cybernetics. Guangzhou, China, Aug 18-21, 2005, Vol 9, pp.5699-703.

[10] J. Li, W. Liu, An agent-based system for multi-project planning and scheduling, Proceedings of the IEEE International Conference on Mechatronics and Automation. Niagara Falls, Ontario, Canada, Jul 29-Aug1, 2005, Vol.2, pp.659-64.

[11] J. Brown, B.J. McCarragher, Maintenance resource allocation using decentralized co-operative control, Proceedings of the Information, Decision and Control. Data and Information Fusion Symposium, Signal Processing and Communications Symposium and Decision and Control Symposium, Adelaide, Australis, Feb 8-10, 1999, pp.41-6.

[12] P.M. Berry, B.Y. Choueiry, L. Friha, Distributed approach to dynamic resource

management based on temporal influence, Intelligent Systems Engineering. 3 (2) (1994) 79-86.

[13] M. Betts, P. Lansley, Construction management and economics: a review of the first ten years, Construction Management and Economics. 11 (4) (1993) 221–45.

[14] R. Pietroforte, T.P. Stefani, ASCE Journal of Construction Engineering and Management: review of years 1983–2000, ASCE Journal of Construction Engineering and Management. 130 (3) (2004) 440–48.

[15] O. Abudayyeh, A.D. Young, E. Jaselskis, Analysis of trends in construction research: 1985–2002, ASCE Journal of Construction Engineering and Management. 130 (3) (2004) 433–39.

[16] J.B. Yang, Developing a knowledge map for construction scheduling using a novel approach, Automation in Construction. 16 (6) (2007) 806-15.

[17] E. Sriprasert, N. Dawood, Multi-constraint information management and visualisation for collaborative planning and control in construction, Itcon. 8 (2003) 341-366.

[18] M. Siddiqui, W.J. O'Brien, A formal mapping-based approach for distributed schedule coordination on projects, Advanced Engineering Informatics. 23 (1) (2009) 104-115.

[19] W. Shen, Q. Hao, H. Mak, J. Neelamkavil, H. Xie, et al., Systems integration and collaboration in architecture, engineering, construction, and facilities management: A review. Advanced Engineering Informatics, DOT.

[20] S.F. Smith, Is scheduling a solved problem? in: G. Kendall, E. Burke, S. Petrovic, M. Gendreau (Eds.), Multidisciplinary Scheduling: Theory and Applications, 2005, pp.3-18.

[21] Q. Hao, Y. Xue, W. Shen, B. Jones, J. Zhu, A decision support system for integrating corrective maintenance, preventive maintenance and condition-based maintenance, Construction Research Congress 2010. Banff, Alberta, Canada, May 8-11, 2010, pp.470-479.

[22] J. Zhu, X. Li, Q. Hao, W. Shen, A New Approach for Resource-Contrained Multi-project Scheduling, Construction Research Congress 2010, Banff, Alberta, Canada, May 8-11, 2010, pp.1084-1093.

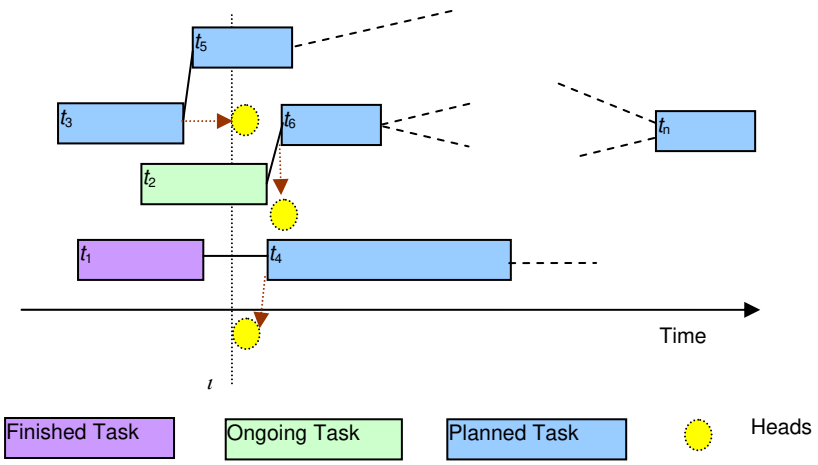

Figure 1: Finding heads for PTN at time ι

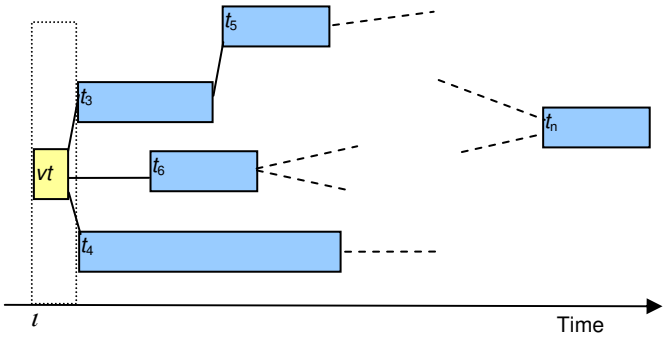

Figure 2: Creating a PTN at time ι

```
INPUT : AT, otr
OUTPUT : CL            // conflict list
BEGIN :
    // el : exclusive task list, rl : resource task list
    CL = ∅;  el = ∅; rl = ∅;
    if (AT ≠ ∅){
        el = {∃t ∈ AT, (t, e_i) ∈ EC, 1 ≤ l ≤ L};
        rl = {∃t ∈ AT, (t, r_k, δ_tk) ∈ RC, 1 ≤ k ≤ K};
    }
    if (el ≠ ∅){
        for(∃t ∈ el){
            cf = ∅;
            for(∃t' ∈ AT, t' ≠ t){
                if (OTR(t, t') ≥ otr)
                    cf.affectedtasklist = cf.affectedtasklist ∪ t';
            }
            if(cf.affectedtasklist ≠ ∅){
                cf.causetask = t;
                cf.status = 'unsolved';
                cf.type = 'exclusive';
            }
            CL = CL ∪ cf;
        }
    }
    // resource conflict detection is samilar with exclusive conflict, but
    // needs to compare the resource constraints of two tasks;
    ...
    // sort conflicts according to the EST of exclusvie task in a conflict.
    if (CL ≠ ∅)
        sortConflictlistByTime(CL);
END.
```

Figure 3: Detect conflicts in a given time window

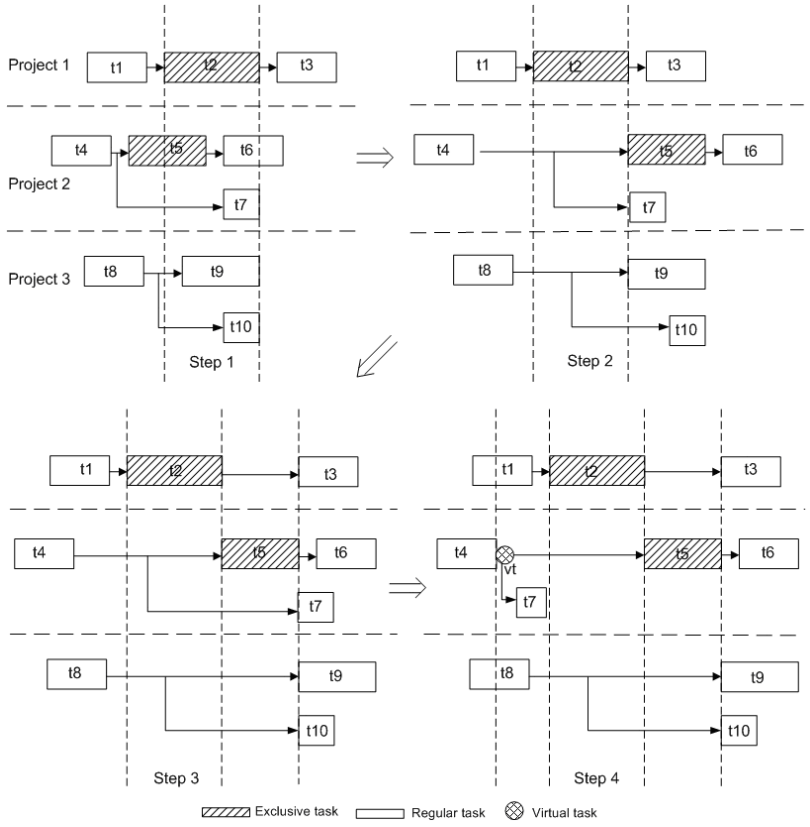

Figure 4: Conflict detecting, resolution and schedule updating

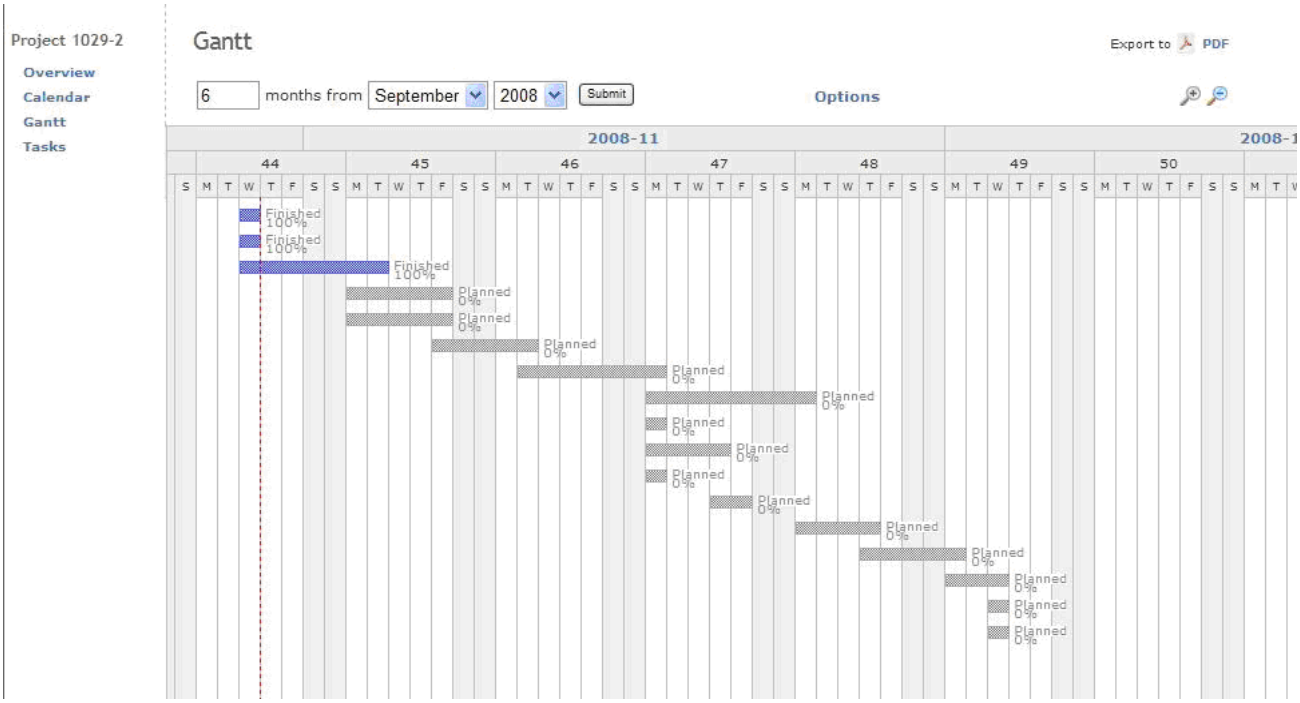

Figure 5: Gantt chart of a small project

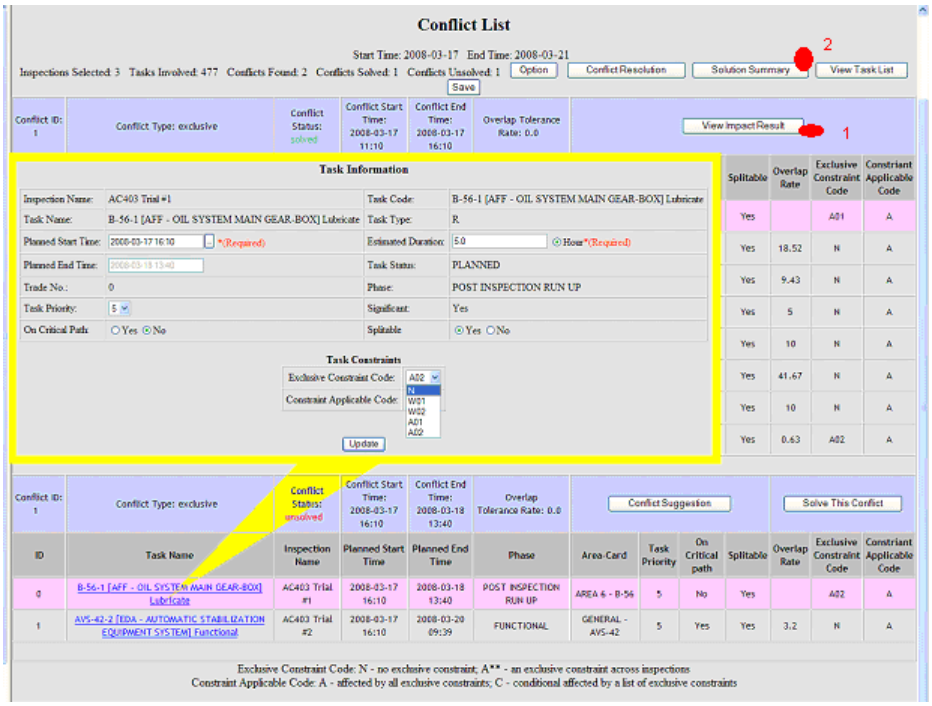

Figure 6: The screen shot of the conflict list

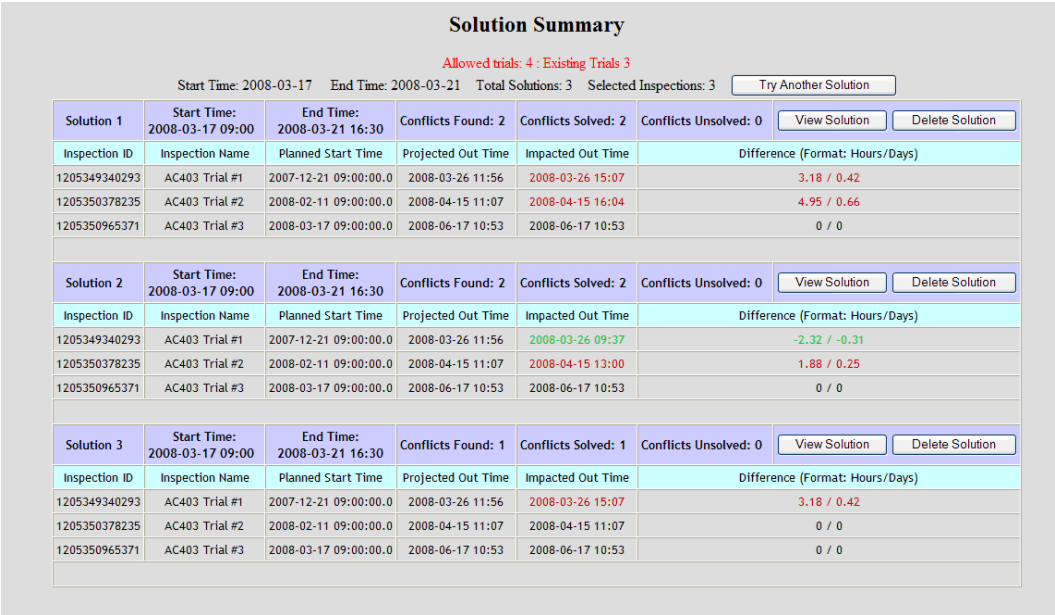

Figure 7: The screen shot of the solution summary