A DYNAMIC CONSTRAINT SOLVING SCHEME FOR SEMI ON-LINE SCHEDULING PROBLEMS

Juyang Zhang Jigui Sun Yonggang Zhang

*College of Computer Science & Technology, Jilin University, 130012, Changchun, P.R.China
Key Laboratory for Symbolic Computation and Knowledge Engineering of Ministry of Education, 130012, Changchun, P.R.China*

abstract: Most of real-life scheduling problems are semi on-line scheduling, where we do not know all the time variables of jobs in advance. Recently, solving such dynamic problems is a hot topic of research in Constraint Programming. In this paper we analyze the constraints in the semi on-line scheduling problems and introduce the concept of monotonic about constraints which appear in general dynamic constraint models. We designed a generic dynamic constraint solving algorithm for solving the scheduling problems. We give an example of its application to semi on-line discrete resource-constrained scheduling problem (ship-loading problem).

key words: Scheduling, Semi On-line, Constraint Solving, Monotonic

1. INTRODUCTION

Constraints are powerful tools for modeling many real-world problems. Constraint programming (CP) is based on the idea of describing the problem declaratively by means of constraints and, consequently, finding a solution satisfying all the constraints, i.e., assigning a value to each unknown from its respective domain. Scheduling is one of the strongest application areas of constraint programming [1]. The reason of such success can be found in a similar character of both scheduling problems and Constraint Satisfaction Problems (CSPs) [2]. In the constraint programming fiend, the constraint models of kinds of scheduling problems are based on CSPs. But the traditional formulation of CSPs is static in sense that all the variables and the constraints are known before we start to solve the problem. Nevertheless,

many real-life scheduling problems such as (semi) on-line scheduling problems do not fit this static constraint model. In those dynamic scheduling problems, we do not know all the time variables of jobs in advance. The new variables are introduced as the schedule progresses, i.e., the original CSPs extend continually. Therefore, we require a more dynamic constraint modeling and solving scheme for those dynamic scheduling problems.

Operations Research (OR) has a long tradition in studying scheduling problems and many successful methods to deal with the dynamic problem were developed there. Recently, solving such dynamic problems is a hot topic of research in CP, however, not a new one. [3] proposed a concept of Dynamic Constraint Satisfaction motivated by problems in configuration. Since then the idea of using activity constraints and dummy variables is used in various concepts. This approach is useful when the number of dummy variables is not very large, i.e., when the number of alternatives is linear rather than exponential. Dynamic Constraint Satisfaction can deal with adding and removing variables and constraints to/from the problem. Addition and retraction can be done in arbitrary order and, thus, the concept must be implemented from scratch and it is hard (or impossible) to extend existing CSP packages to support it. In this paper, we designed a straightforward generic technique for making monotonic constraints in semi on-line scheduling problems dynamic. This scheme is easy to implement in the current scheduling systems, as it does not require a change of paradigm from CSP to Dynamic CSP and it can exploit directly the existing filtering algorithms for constraints.

In section 2, we first describe the semi on-line scheduling problem. The details about the generic dynamic constraint solving scheme including the concept of monotonic constraints and the dynamic constraint solving algorithm *Dyna-Solver* are showed in Section 3. In section 4, we give an example of its application to on-line discrete resource-constrained scheduling problem (ship-loading problem). Finally, we draw a conclusion on this new scheme.

2. SEMI ON-LINE SCHEDULING PROBLEM

Considering the information provided the task, scheduling falls generally into two categories^[4], off-line and on-line. In the off-line scheduling all the information of tasks is known in advance, and this makes it possible to perform scheduling based on these given information. However in the on-line cases, the following two assumptions were made: (1) Information is given by sequence, i.e. the information about task T_{j+1} can only be

announced after the scheduling of the tasks $T_1, \dots, T_j$; (2) No changes are allowed to scheduled tasks, i.e. if a task is sent to some machine it is prohibited to re-schedule the task.

As described above, both off-line and on-line scheduling are idealized cases in terms of the quantity of knowledge provided. In case of off-line job scheduling, all the information required is given, so that an algorithm of high performance may be obtained by conducting operations on data, such as sorting, operating, and comparing. Considering an on-line scheduling no information, including the release-time, the amount of need and supply, and the number of the rest tasks, can be obtained in advance; the schedules made for former tasks are not allowed to be changed. In practice, however, few cases were found falling into these two idealized categories. Although the overall knowledge about tasks waiting to be scheduled may not be fully provided, it is generally possible for schedulers to have some information in advance, and/or to re-schedule the former tasks with higher degree of freedom than that in conventional on-line scheduling. Thus scheduling problems in practice will be solved by overcoming more difficulties than idealized off-line ones, however less effort might be taken to achieve a relative optimized algorithm than on-line scheduling. These scheduling problems, where the above two assumptions for on-line sorting can not fully apply, are so-called semi on-line scheduling problems. First reported in 1996, the studies based on semi on-line concept have been conducted in both practical and theoretical research fields for the last decade. Models being developed based on this concept, a new branch of scheduling was recognized as semi-on-line scheduling.

3. DYNAMISATION CONSTRAINT SOLVING SCHEME

In the semi on-line scheduling problem, we do not know all the time variables of tasks in advance. The new variables are introduced as the schedule progresses, i.e., the original constraint satisfaction problem extend continually. Consequently, we cannot post a constraint $C(X)$ until we know all the variables X . Nevertheless, if the constraint is monotonic we can do better by posting a constraint over the known variables and when a new variable (or a variable set Y) arrives we can post a new constraint extended by the new variable and deactivate the old constraint. This is the basic idea behind a generic dynamisation scheme that we will describe in details now.

3.1 Dynamic constraint modeling

A *Constraint Satisfaction Problem*^[5] $P=(X,D,C)$ is defined as a set of variables $X=\{x_1, \dots, x_n\}$, a set of domains $D=\{D(x_1), \dots, D(x_n)\}$, where $D(x_i)$ is a finite set of possible values for variable x_i , and a set of constraints $C=\{c_1, \dots, c_m\}$. A *constraint* c_i on the ordered set of variables $X(c_i)=(x_{i1}, \dots, x_{ik})$ is a subset of the Cartesian product $D(x_{i1}) \times \dots \times D(x_{ik})$, it specifies the allowed combinations of values for the variables $x_{i1}, \dots, x_{ik}$. $|X(c_i)|$ is called arity of the constraint c_i .

Assume that $(v_1, \dots, v_n) \downarrow (x_{i1}, \dots, x_{ik})$ is a standard projection operator, i.e., a projection of the tuple $(v_1, \dots, v_n)$ to the variables $(x_{i1}, \dots, x_{ik})$: $(v_1, \dots, v_n) \downarrow (x_{i1}, \dots, x_{ik}) = (v_{i1}, \dots, v_{ik})$. We call a complete instantiation $(v_1, \dots, v_n)$ of the variables, such that $\forall i=1, \dots, n \ v_i \in D(x_i)$ and $\forall j=1, \dots, m \ (v_1, \dots, v_n) \downarrow X(c_j) \in c_j$, a *valid tuple* or *solution*, i.e., the valid tuple is a complete instantiation of the variables satisfying all the constraints. $Sol(P)$ is a set of all valid tuples, we call it a *solution set* of the constraint satisfaction problem P .

But the traditional formulation of a *CSP* is static in sense that all the variables and the constraints are known before we start to solve the problem. Nevertheless, the semi on-line scheduling problem does not fit this static constraint model and that require a more dynamic constraint modeling and solving scheme. Here we introduce the concept of monotonicity to tackle this problem. Monotonicity means that extending a constraint, i.e., adding new variables to the constraint, does not add new solutions. In the semi on-line problem, we can find there are some constraints which have this speciality. The following is the definition about monotonicity.

Definition 1: Problem $P'=(X',D',C')$ is called an *extension of the problem* $P=(X,D,C)$, if $X \subseteq X'$, $C \subseteq C'$ and $\forall x \in X \ D(x)=D'(x)$. Briefly speaking, we extend the original problem P to problem P' if we add new variables and constraints to it.

Definition 2: *Constraint extension is monotonic*, if P' is an extension of P , then $Sol(P') \downarrow X \subseteq Sol(P)$, where X is a set of variables in P , i.e., every solution of the extended problem is also a solution of the original problem.

Let $C(X)$ be a function that maps arbitrary finite ordered set of variables to a conjunction of constraints, which forms a sub-problem in the constraint satisfaction problem. Note, that the constraints in $C(x_1, \dots, x_k)$ may contain variables other than $x_1, \dots, x_k$ but these variables are local/hidden there, i.e.,

they do not appear in the rest of the problem. If we have a constraint conjunction $C(X)$, we say that $C(Y)$ is its extension if $X \subseteq Y$.

Definition 3: Let $Sol(C(X), D(X))$ be a solution set for the constraint $C(X)$ and a set of variables X with domains $D(X)$. A *constraint solving scheme/algorithm* is *monotonic*, if $Sol(C(X \cup Y), D(X \cup Y)) \downarrow X \subseteq Sol(C(X), D(X))$ for arbitrary sets X and Y of variables.

Generally speaking, monotony of constraint extension means that extending a constraint, i.e., adding new variables to the constraint, does not add new solutions. And monotony of constraint solving scheme means that if solving starts with larger domains then it does not achieve better pruning of the domains than when starting from smaller domains.

3.2 Algorithm Dyna-Solver

Our objective is to design a generic dynamisation constraint solving algorithm that allows extending the set of variables in the constraint scheme on fly while preserving consistency achieved by the solving algorithm for the static constraint. The dynamisation algorithm *Dyna-Solver* we designed is a dynamic extension of the static constraint solving algorithm *CB-Solver*^[6], which maps a constraint $C(X)$, domains $D(X)$ and a new set of variables Y to domains $D'(X \cup Y)$ such that $Sol(G(X \cup Y), D(X \cup Y)) \subseteq D'(X \cup Y) \subseteq CB-Solver(G(X \cup Y), D(X \cup Y))$.

Algorithm *Propagation* for the constraint $C(X)$, which is wrapped in a trigger procedure p is a part of *CB-Solver*. In particular, procedure $p(x_1, \dots, x_n)$ calls the filtering algorithm for the constraint $C(X)$ every time a domain of any variable x_i is changed. We will use this procedure as the basic interface to the filtering algorithm for $C(X)$. Our algorithm exploits the monotony property of the constraint in semi on-line scheduling problem. In particular, we can start with the previous solution (domains) to get a solution (restricted domains) of the extended constraint.

Compared with the static algorithm *CB-Solver*, *Dyna-Solver* has three improvements. The algorithm first handles the set of the extending variables (step 1). Variables can be removed from the constraint only upon backtracking, i.e. LIFO (last-in first-out) mechanism is applied for adding/removing variables to/from the constraint. Moreover, if we add a set of variables, the whole set is removed together upon backtracking. LIFO mechanism for adding/removing the variables may seem like a restriction

but it simplifies significantly the implementation. Then, it deactivates the current propagation algorithm (step 2), note that it does not mean removing the constraint from the constraint store, it is just stopping propagation through it (the internal data structures are kept). Then, it saves domains of the variables to the stack (step 3), typically, the underlying constraint engine saves such information; we highlight it just to illustrate the track of the algorithm. The following is the detail of *Dyna-Solver*.

Dyna-Solver($C(X_1, \dots, X_k), X_{k+1}$)

- 1) Check the length of the variable stack, i.e., judge the number of the variables is changed or not
- 2) Deactivate $p(X_1, \dots, X_k)$, i.e., push its internal data structures to the stack and stop propagation through this constraint (if $k \geq 1$, otherwise do nothing)
- 3) Push domains of $X_1, \dots, X_k, X_{k+1}$ to the variable stack
- 4) Judge whether there is a variable in the variable stack. If no, skip to step(1)
- 5) Judge whether the domain of the variable is null. If yes, skip to step(9)
- 6) Select the variable that has the smallest domain and the most constraints to be the current variable
- 7) Judge the domain of current variable is null or not. If not, skip to step(9)
- 8) Check the constraint stack of current variable, i.e., judge whether any constraint propagation events happen. If the constraints are consistency (use the trigger procedure p to judge) then skip to step (4), else skip to step (5)
- 9) Backtrack. If success then skip to step (5), else terminate

When removing a variable upon backtracking, we simply reverse algorithm *Dyna-Solver*. It means: removing the extended constraint from the constraint store (now, it is real removing, not just deactivation), restoring domains of the variables (this is usually done automatically when the extended constraint is removed from the constraint store), and activating the former constraint again.

4. AN EXPERIMENTAL STUDY OF DYNA-SOLVER

Shop-loading scheduling is one kind of discrete resource-constrained scheduling problems, in which tasks require a discrete resource of capacity strictly greater than 1. In this context, two tasks which require the same

resource may overlap in time, but the total capacity required at time t must not exceed the capacity available at time t . The problem is solved by assigning precise start and end times to each task in the schedule. The propagation of resource constraints is exploited to prune the search tree and to guarantee that the available capacity is never exceeded.

In the general/off-line shop-loading problem, the number of the tasks, the duration of tasks and the required amounts of resource capacity are fixed once and for all. The capacity of the resource (h) is also fixed. The goal is to minimize the makespan. Formally: task set $T = \{ T_1, T_2, \dots, T_n \}$, duration set $Dur = \{ Dur_1, Dur_2, \dots, Dur_n \}$, required amounts of resource capacity set $T(R) = \{ T_1(R), T_2(R), \dots, T_n(R) \}$. A shop-loading scheduling problem can be encoded efficiently as a CSP:

- The time variable set $X = \{ st_1, st_2, \dots, st_n \}$, time variable st_i is associated with each task T_i ; it represents the start time of T_i .
- The domain D_i of st_i is initialized with $[0, \sum_{i=1}^n Dur_i]$.
- The constraint Set C:
Temporal precedence constraint: $Successors(T_i) = T_j$ means that $st_j \geq st_i + Dur_i$; Resource cumulative constraint: $Cumulative((st_1, \dots, st_n), (Dur_1, \dots, Dur_n), (T_1(R), \dots, T_n(R)), h)$. The constraint enforces the condition $\sum_{i=1}^n T_i(R) \leq h$, all $t (st_i \leq t \leq st_i + Dur_i)$.

Consider the problem, which consists of 13 tasks ($n=13$) subject to precedence constraints, as indicated in the table1. The capacity of the resource R is 8 ($h=8$).

Table 1. Example of Ship-loading

T_j	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}
Successors	$T_2 T_4$	T_3	$T_5 T_7$	T_5	T_6	T_8	T_8	T_9	T_{10}	$T_{11} T_{12}$	-	T_{13}	-
Dur_j	3	4	4	6	5	2	3	4	3	2	3	2	1
$R(T_j)$	4	4	3	4	5	5	4	3	4	8	4	5	4

Considering this example in off-line case, we solve it by using *CB-Solver*. The schedule results are displayed in the figure1. makespan=35.

Figure 1. Schedule Results in Off-line Case

Considering this example in semi on-line case, we assume a new task is introduced at time $t=20$. We name the new task T_{14} ($Dur_{14}=3$, $T_{14}(R)=1$). The adding time variable set $Y=\{st_{14}\}$. Extension constraints: $Successors(T_8)=T_{14}$, i.e., $V_{14} \geq V_8 + Dur_8$ and $Cumulative(X \cup Y, Dur \cup Dur_{14}, T(R) \cup T_{14}(R), h)$. The constraint solving scheme *Dyna-Solver* ($Cumulative(X, Dur, T(R), h), Y$) is monotonic because there are no local variables in the *Cumulative* constraint. The schedule results are displayed in the figure2.

Figure 2. Schedule Results in Semi On-line Case

The results show that the adding variable st_{14} in the constraint *Cumulative* does not bring the new result to the original solution. So it is not essential for us to re-schedule the new problem because of the extension of monotonic constraints.

5. CONCLUSIONS

In the Operation Research field, the semi on-line scheduling problems are studied for a long time. But in the constraint programming field, the related research just begins. This paper presents the dynamisation of monotonic constraints solving in the semi on-line scheduling problems. We designed a generic dynamic constraint solving scheme, which adapts to arbitrary monotonic constraint appears in the semi on-line scheduling problem. In the highly dynamic problems, our approach is even more beneficial. However,

there are also many non-monotonic constraints. These constraints cannot be dynamized using the scheme we proposed. The main obstacle here is that adding new variables to a non-monotonic constraint might add new solutions to the constraint. Consequently, to design a constraint solving scheme that can be used to solve real-life dynamic scheduling problems is also the biggest challenge of current constraint programming.

ACKNOWLEDGEMENTS

This paper is supported by the National Natural Science Foundation Key Project of China (Grant No.60273080, 60473003), the Program for New Century Excellent Talents in University and the Outstanding Youth Foundation of Jilin Province of China (Grant No.20030107).

REFERENCES

- [1] Wallace, M.: Applying Constraints for Scheduling, in Constraint Programming, Mayoh B. and Penjaak J.(eds.), NATO ASI Series, Springer Verlag (1994)
- [2] Tsang, E. P. K.: Foundations of Constraint Satisfaction, San Diego, Calif.: Academic (1993) 53-63
- [3] Mittal, S., Falkemhainer, B.: Dynamic Constraint Satisfaction Problems, In Proceedings of AAAI-90 (1990) 25-32
- [4] Sgall J.: On-line scheduling. In: Fiat, A. and Woeginger, G. eds. , On-line Algorithms: The State of Art, Lecture Notes in Computer Sciengces, 1442. Berlin: Springer-Verlag (1998)
- [5] Tsang, E. P. K, Foundations of Constraint Satisfaction. San Diego, Calif.: Academic (1993) 5-10
- [6] Juyang Zhang, Xin Li, Jigui Sun: Research On Constraint-based Scheduling and Its Implementation, Proceeding of CNCC'03, Beijing, The Tsinghua University Press (2003) 80-85