

Trustworthy Dynamic Scheduling based on Constraint Solving Scheme

Juyang Zhang, Yixiang Chen
Software Engineering Institution, East China Normal University
Shanghai, P.R.China
jyzhang@sei.ecnu.edu.cn, yxchen@sei.ecnu.edu.cn
doi: 10.4156/ijipm.vol1.issue2.9

Abstract

*Most of real-life scheduling problems are dynamic, where we do not know all the time variables of jobs in advance. In such uncertain environment, trustworthy scheduling effectively is a hot topic. In this paper we model the constraints in the dynamic scheduling problems by using CSP model. Moreover, we propose model the time constraint variables in logic way. Based on the constraint models, a constraint solving algorithm *Resol* for solving the scheduling problems is designed. We prove that this algorithm is sound for solving the scheduling problems. The experimental study proves that the trustworthy of the schedule system can be promoted by applying the algorithm.*

Keywords: *Trustworthy, Dynamic Scheduling, CSP, Constraint Solving*

1. Introduction

Constraints are powerful tools for modeling many real-world problems. 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 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). The trustworthy of schedule become more and more important.

Therefore we require a more trusted scheduling 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. [4] analyzed the problems behind planning and scheduling in complex process environments and proposed to enhance the traditional schedulers by planning capabilities to solve these problems. [5] introduced the concept of monotonic constraint and designed a dynamic constraint solving algorithm for solving the dynamic scheduling problems.

In this paper we attempted to promote the trustworthy of the dynamic scheduling problems by using the constraint solving framework. In section 2, we first describe the dynamic scheduling problems and the discrete variable constraint modeling in logic way. The details about the constraint solving algorithm *Resol* are showed in Section 3 and we also give the proof about the sound of the algorithm in this Section. In section 4, an experimental study of algorithm *Resol* is given. Finally, we draw a conclusion on this new scheme.

2. Constraint modeling

2.1. CSP model

A *Constraint Satisfaction Problem* $P=(X,D,C)$ 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 $(v_1,\dots,v_n) \in (x_{i1},\dots,x_{ik})$ 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}) : $(v_1,\dots,v_n) \in (x_{i1},\dots,x_{ik}) = (v_{i1},\dots,v_{ik})$. We call a complete instantiation (v_1,\dots,v_n) of the variables, such that $v_i \in D(x_i)$ and $j=1,\dots,m$, $(v_1,\dots,v_n) \in X(c_j)$ 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 .

Based on the CSP model, the general method of modeling dynamic constraints in it is to list all the assignment combinations of variables directly. We propose describe the dynamic constraint by using the formula of discrete-variable logic. In the framework of discrete-variable logic, the dynamic scheduling problem can be solved in the method of multivalent resolution on finite sets.

In the dynamic scheduling problems, we do not know all the time variables of jobs in advance. Consequently, we cannot post a constraint set $C(X)$ (multivalent clauses) until we know all the variables X . Nevertheless, if the constraint is monotonic^[5] 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.

2.2. Time discrete variable model

We name the time variables in dynamic scheduling multivalent variables, which have the finite and discrete domains. An elementary extension of propositional logic can be developed for multivalent variables. In propositional logic, the primitive unanalyzed terms are atomic formulas y_j . The analysis can be carried slightly deeper by supposing that atomic propositions are themselves predicates that say something about discrete variables x_1,\dots,x_n , which can be regarded as the time variable in RCSP. For instance, a predicate may have the values x_j can assume. Special cases would be $x_j=v$ and $x_j \neq v$, where v is a constant. A number of useful predicates can be defined in terms of more primitive notation, just as equivalence \equiv and implication \supset are defined in terms of \vee and \neg in propositional logic.

The resulting logic is still bivalent in that propositions have one of two truth values. The variables, however, are multivalent.

Whereas a limited repertory of connectives appear to be useful in propositional logic, multivalued variables multiply the possibilities. The all-different, element, distribute, and cumulative predicates have proved especially useful. The idea of a logical clause is also readily generalized.

- **Formulas and Semantics**

The atomic propositions y_j of propositional logic are replaced with predicates $P(x) = P(x_1,\dots, x_n)$ in discrete variable logic. Predicates can be combined with logical connectives in the same way as logical propositions. one primitive predicate will be sufficient to define all others, namely $P(x)=(x_j \in X_j)$ for $X \subset D_j$.

The semantics are slightly different than in propositional logic. In the latter, the meaning of a molecular formula is given by the Boolean function it represents. In discrete logic, a formula's meaning is given by a truth function $f(x)$ of the discrete variables $x = (x_1,\dots,x_n)$, where each $x_j \in D_j$. In particular, each predicate is defined by the function $f(x)$ it represents. For example, the function $f(x)$ for $x_j \in Y_j$ takes the value 1 if the value assigned x_j belongs to X . Once the truth values of the predicates are determined, the truth values of the formulas containing them are computed in the normal propositional way.

- **Multivalent Clauses**

Multivalent clauses are a straightforward generalization of propositional clauses and are completely expressive in an analogous sense.

A multivalent clause has the form

$$\bigvee_{j=1}^m (x_j \in X_j) \quad (1)$$

where each $X_j \subset D_j$. If X_j is empty, the term $(x_j \in X_j)$ can be omitted from (2), but it is convenient to suppose here that (2) contains a term for each j . If $X_j = D_j$ for some j , then (2) is a tautology. Note that the literals of a multivalent clause contain no negations. This brings no loss of generality, since $\neg(x_j \in X_j)$ can be written $x_j \in D_j \setminus X_j$.

Any truth function $f(x) = f(x_1, \dots, x_n)$ can be expressed as a conjunction of multivalent clauses. This is done simply by ruling out the values of y for which $f(x) = 0$. Thus, if $f(x) = 0$ for $x = v^1, \dots, v^k$, then $f(x)$ is represented by the formula

$$\bigwedge_{i=1}^k \bigwedge_{j=1}^n (x_j \neq v_j^i) \quad (2)$$

which can be formally written as a multivalent clause:

$$\bigwedge_{i=1}^k \bigwedge_{j=1}^n (x_j \in D_j \setminus \{v_j^i\}) \quad (3)$$

Because any constraint over finite domains represents such a function $f(y)$, it is equivalent to a finite set of multivalent clauses.

One multivalent clause $\bigvee_j (x_j \in X_{1j})$ implies another $\bigvee_j (x_j \in X_{2j})$ if and only if the one absorbs the other; that is, $X_{1j} \subset X_{2j}$ for each j . Equivalent multivalent clauses are identical. Prime implications are defined precisely as for classical clauses.

Any formula of discrete logic can be converted to a conjunction of multivalent clauses by using De Morgan's laws, distribution, double negation, and the fact that $\neg(x_j \in X_j)$ means $(x_j \in D_j \setminus X_j)$.

3. Constraint solving algorithm *Resol*

Resolution is easily extended to the logic of discrete variables. Unit resolution also has an analog. Resolution plays the same role in computing projections as it does in propositional logic. A unit resolution algorithm *Resol* for multivalent clauses is very similar to classical unit resolution. When a clause C_k becomes a unit clause $x_j \in X_{kj}$, the domain of x_j can contain only elements that occur in X_{kj} . The clause C_i is deleted and the clauses containing x_j adjusted accordingly.

To speed processing, the algorithm keeps track of which literals remain in each clause. Thus χ_i contains the nonempty sets X_{ij} . It also maintains a list S_j of the clauses that still contain x_j ; that is, the clauses C_i for which X_{ij} is nonempty. Whenever X_{kj} in some clause C_k becomes empty, X_{kj} is removed from χ_k . If this makes C_k a unit clause, then every X_{ij} in the constraint set must be updated so that it lies in X_{kj} . The list S_j makes it possible to locate quickly the X_{ij} 's that might be affected. The clause C_k is deleted from the problem and removed from S_j .

Soundness means that no solution is removed by solving. The following proof guarantees that the algorithm we designed is soundness.

Proof:

1. $D'(X) \subseteq D(X)$
2. $Resol(C(X \cup Y), D'(X) \times D(Y))$
 $\subseteq Resol(C(X \cup Y), D(X \cup Y))$
monotony of *Resol*
 $+D'(X) \times D(Y) \subseteq D(X) \times D(Y)$
 $= D(X \cup Y)$
3. $Resol(C(X), D(X)) \subseteq D'(X)$
4. $\forall s \in Resol(C(X \cup Y), D(X \cup Y))$

5. $s \downarrow X \in Resol(C(X), D(X))$	monotony of C
6. $s \downarrow X \in D'(X)$	3+5
7. $s \in D'(X) \times D(Y)$	
8. $s \in Resol(C(X \cup Y), D'(X) \times D(Y))$	4+7+feature of ReSol
9. $Resol(C(X \cup Y), D(X \cup Y)) \subseteq Resol(C(X \cup Y), D'(X) \times D(Y))$	4-8
10. $Resol(C(X \cup Y), D'(X) \times D(Y)) \subseteq Resol(C(X \cup Y), D'(X) \times D(Y))$	
11. $Resol(C(X \cup Y), D(X \cup Y)) \subseteq Resol(C(X \cup Y), D'(X) \times D(Y))$	9+10

A unit resolution algorithm <i>Resol</i>
<p>Let S be a set $\{C_i \mid i \in I\}$ of multivalent clauses,</p> <p>where each C_i has the form $\bigvee_{j=1}^n (x_j \in X_{ij})$</p> <p>For all $i \in I$ and $j=1, \dots, n$ let $X_{ij} = X_{ij} \cap D_{x_j}$, where D_{x_j} is the current domain of x_j.</p> <p>For $i \in I$:</p> <p>Let χ_i be the collection of nonempty sets X_{ij}.</p> <p>If $\chi_i = \emptyset$ then stop; S is unsatisfiable.</p> <p>For $j=1, \dots, n$ let $S_j = \{C_i \mid X_{ij} \in \chi_i\}$.</p> <p>Let U be the set of pairs (C_k, j_k) for which C_k is a unit clauses; that is, $\chi_k = \{X_{kj_k}\}$.</p> <p>While U is nonempty:</p> <p>Remove some pair (C_k, j_k) from U.</p> <p>Remove C_k from S_{j_k}.</p> <p>For all $C_i \in S_{j_k}$:</p> <p>Let $X_{ij_k} = X_{ij_k} \cap X_{kj_k}$.</p> <p>If $X_{ij_k} = \emptyset$ then:</p> <p>Remove X_{ij_k} from χ_i.</p> <p>If $\chi_i = \emptyset$ then stop; S is unsatisfiable.</p> <p>If $\chi_i = 1$ then add C_i to U.</p>

4. An experimental study of algorithm *Resol*

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 represent 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 table 1. The capacity of the resource R is 8 ($h=8$).

Table 1. Example of Ship-loading

T_i	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_i	3	4	4	6	5	2	3	4	3	2	3	2	1
$R(T_i)$	4	4	3	4	5	5	4	3	4	8	4	5	4

Considering this example in dynamic 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 schedule results by using the constraint solving algorithm $Resol(Cumulative(X, Dur, T(R), h), Y)$ are displayed in the figure 1.

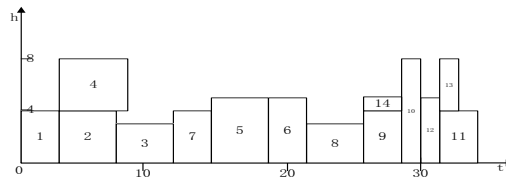


Figure 1. Schedule Results in Dynamic Case

5. Conclusions

This paper presents the dynamisation of scheduling problems in the trustworthy system. We model the multivalent time variables in logic way and design a constraint solving algorithm, which adapts to arbitrary monotonic constraint appears in the scheduling problem. In the highly dynamic problems, our approach is even more beneficial. The soundness of the algorithm is proofed. This theoretical result provides the guaranty for designing trustworthy schedule system. To design a real-life schedule system by using this constraint solving scheme is our next research work.

6. 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, pp.53-63, 1993.

- [3] Mittal, S., Falkemhainer, B.: Dynamic Constraint Satisfaction Problems, In Proceedings of AAAI-90, pp.25-32, 1990.
- [4] Barták, R.: Dynamic Constraint Models for Planning and Scheduling Problems, In New Trends in Constraints. LNAI 1865. Springer Verlag, pp.237-255, 2000.
- [5] ZHANG Juyang, SUN Jigui, YANG Qingyun: A Constraint Solving Algorithm for Semi On-line Scheduling Problem, Acta Automatica Sinica, vol. 33, no. 7, pp.765-767, 2007.