A Logic-based Branch and Bound Algorithm for Resource Constrained Scheduling Problem

Juyang Zhang, Jigui Sun, Qingyun Yang,

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

Summary

Resource-constrained scheduling problem is one kind of typical real-life discrete optimization problems, which is one of the strongest application areas of constraint programming. We design a new logic-based method for solving the resource-constrained scheduling problem. In this paper, we propose a way of describing those constraints with the discrete-variable logic formula. Based on this model, a logic-based branch and bound algorithm is designed for solving the discrete variables' domain. Comparisons with other constraint handling approaches and related literature clearly show that our approach can describe the constraints in the high level and solve the resource-constrained scheduling problem in the logic framework.

Key words:
Scheduling; Logic; Branch and Bound; Constraint Programming

Introduction

Constraints are powerful tools for modeling many real-world problems. Constraint programming (CP) [1] 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. CP has a big advantage over other frameworks in declarative modeling capabilities. The modeling capabilities of CP are really fascinating and the constraint models are very close to the description of real-life problems. This simplifies the maintenance of the models as well as the introduction of domain dependent heuristics necessary to solve large-scale problems such as Resource-Constrained Scheduling Problem (RCSP), which is one of the strongest application areas of CP [2]. The reason of such success can be found in a similar character of both scheduling problem and Constraint Satisfaction Problem (CSP) [3].
Our system "Mingyue" CB-Scheduler [4, 5] is one kink of CP toolkits, which embed constraints in the object-oriented programming language C++. Other constraint-solving toolkits are also quite popular, such as ILOG solver and scheduler. However, in the typical CP toolkit, constraint modeling is to list all the assignment combinations of variables directly in the CSP model. In this paper, we propose describe and solve the RCSP by using formula of discrete-variable logic and logic-basic method, which is more general and brief.
In this paper we omit the description of the system "Mingyue" CB-Scheduler and emphasize the constraint solving method in the system. In section 2, we first describe the definition of RCSP and then we present a logic-based CSP model of RCSP. The details about modeling constraints in the discrete-variable logic way are showed in section 3. In the section 4, we give the logic-based branch and bound algorithm for solving the constraints in RCSP. Finally, we draw a conclusion on this new method.

2. Logic-based constraint modeling in RCSP

Partially based upon the scheduling problems we encountered in the industry, we define a generic (and necessarily incomplete) typology of resource-constrained scheduling as follows:
Given are a set of n tasks T={T1, …, Tn} and a set of m resources R={R1, …, Rm}. Each task Ti has its certain time-window [ri, di] (ri denotes the release-date, di denotes the due-date) and needs some amount of resource Rj throughout their execution. Each resource Ri has its certain capacity C. Several tasks can be processed simultaneously, provided that the total resource consumption does not exceed capacity C at any time. The objective is to minimize makespan (the finish time of the last finished task). A solution schedule is a set of integer execution times for each task so that all the temporal and resource constraints are satisfied.
A RCSP can be encoded efficiently as a CSP: two variables, sti and fti, are associated with each task Ti; they represent the start time and the finish time of Ti. The smallest values in the domains (D) of sti and fti are called the release-date and the earliest finish time of Ti (ri and efti). Similarly, the greatest values in the domains of sti and fti are called the latest start time and the due-date of Ti (lsti and di). The processing time of the task is an additional variable pti, that is constrained to be lower than

Manuscript received February 25, 2006.
Manuscript revised February 28 , 2006.

or equal to the difference between the end and the start times of the task (most often, processing time is known and bound to a value duri).

A basic logic-based model of RCSP may be defined in which models have the following general form:

minimize: $C_{max} = \max\{fti \mid i=1,\ldots,n\}$

subject to

$ri \le sti \le di\text{-}pti$	(checkable constraint)
$ri+pti \le fti \le di$	(soluble constraint)
cumulative((st1,..stn),(pt1,…,ptn),(c1,…,cn),C)	(defined constraint)
$fti \in Di$	(solution variable)
$sti \in Di$	(search variable)

The key idea for constraint modeling in RCSP is the distinction of soluble constraints from checkable constraints. Soluble constraints are those structure makes them suitable for fast solution by optimization methods. Checkable constraints are less amenable to efficient optimization, bur one can easily check whether a given solution satisfies them, including logical propositions and the global constraints.

Because a checkable constraint can be evaluated only after a solution is specified, values for its variables must be enumerated. One might refer to them as search variables. The overall structure of the solution algorithm is therefore to branch on the search variables. Branching is accomplished by fixing values of variables or partitioning their domains into smaller sets. The soluble constraints might be inequalities that are suitable for linear or convex nonlinear programming. It is not necessary to branch to obtain feasible or optimal values for their variables. Their variables can therefore be called solution variables.

Defined constraint cumulative((st1,..stn), (pt1,…,ptn), (c1,…,cn), C), where the c1,…,cn represent the amount of resource consumed by task. The constraint enforces the condition

$$\sum_{\substack{j \\ sti \le t \le sti+duri}} c_j \le C \text{, all t}$$

(1)

3. Multivalent variable and multivalent resolution

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

3.1 Formulas of Discrete-Variable Logic

We name the time variables (sti and fti) in RCSP 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 $x_1,\ldots,x_n$, 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 $P(x) = P(x_1,\ldots, x_n)$ 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 $X \subset Dj$.

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 $x = (x_1,\ldots x_n)$, 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

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

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 $x_j \in D_j \backslash X_j$.
Any truth function $f(x) = f(x_1,\ldots,x_n)$ 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 $x = v^1,\ldots, v^k$, then f(x) is represented by the formula

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

which can be formally written as a multivalent clause:

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

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 $\vee_j (x_j \in X_{1j})$ implies another $\vee_j (x_j \in X_{2j})$ 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 $(x_j \in D_j \backslash X_j)$.

3.2 Multivalent Resolution

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. In [7], we designed a unit resolution algorithm for multivalent clauses. It 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.

4. A logic-based branch-and-bound algorithm

A generic logic-based branch-and-bound algorithm appears in Table 1. It characterizes each node of the search tree with a triple (CC, SC, DC, D). CC and SC are sets of checkable and soluble constraints, respectively. G is a set of defined constraints. D constrains of the current domains $D_1,\ldots,D_n$ of the search variables st1,…,stn. If the branching process has fixed a variable stj to some value v, this is reflected by the fact that its domain Dj is the singleton {v}. Multivalent resolution[7] can also reduce the domains.
The search is controlled by the node selection procedure, which selects a node to explore from the set A of active nodes. Active nodes are those that have been created but are yet unsolved. They are created when the algorithm branches on a variable.
When an active node is selected, multivalent resolution is applied to CC and the checkable constraints in DC. This may add checkable and/or soluble constraints, including no-goods, and it may reduce domains. If some domain is reduced to the empty set or infeasibility is otherwise detected then problem is unsatisfiable, and the search backtracks.
If no unsatisfiablility is detected, a relaxation consisting of a set R of soluble constraints is formulated. The objective is to minimize the C_{max}. The relaxation receives constraints from consequents of conditionals whose antecedents are true, and from relaxations of defined constraints. Solution of the relaxation can generate no-goods as well as information that aids branching decisions.
If the optimal value C_{max} of the problem is greater than or equal to than the value $\overline{C_{\max}}$ of the incumbent solution, then there is no point in further consideration of the current node, and the search backtracks. Otherwise an attempt is made to select values st1,…stn to obtain a feasible solution($\overline{C_{\max}}$,st). If this fails, a variable is selected for branching, using guidelines that were generated in the course of processing the current node. The algorithm continues until the set A of active nodes is exhausted.

Table. 1 A logic-based branch-and-bound algorithm in RCSP

```
Let CC= {ri ≤ sti≤ di-pti | i∈1,..n}, SC= {ri+pti ≤ fti ≤ di
| i∈1,…,n}, DC= {cumulative((sti, pti, ci,,C)| i∈1,…n}
Add to DC the defined constraint in CC and SC.
Let D= {D1,…,Dn} be the domains of st1,…,stn.
Let $\overline{C_{\max}}$ be the upper bound on the optimal value, initially ∞.
Let A be the set of active nodes, initially with A= {(CC,SC,DC,D)}.
While A is nonempty:
  Node selection: Remove a tuple (CC,SC,DC,D) from A.
  Perform Multivalent Resolution.
  If no infeasibility is detected then
    Perform Relaxation.
    If z<$\overline{C_{\max}}$ then
      If some |D_j|≠ 1 then perform Completion.
      If each |D_j|= 1 then let (ft*, D*)= ($\overline{ft}$, D) and $\overline{C_{\max}}$ = c $\overline{ft}$;
      Else perform Branching.
If Cmax*<∞ then (ft*, D*) is an optimal solution;
Else the problem is infeasible.

Procedure Multivalent Resolution.
  Apply inference algorithms as desired to CC and DC, possibly changing CC, SC, DC and D.
  Let B the set of branching guidelines generated.

Procedure Relaxation.
  Generate soluble relaxations as desired for formulas in CC and DC add their constraints to SC.
  Let R contain the constraints in SC.
  Repeat as desired or until R becomes infeasible:
    Optimization: Let $\overline{ft}$ minimize Cmax subject to R.
      If R is infeasible set Cmax= ∞.
      Else add separating cuts to R.
  Add branching guidelines to B.
  Add generated constraints and no-goods to CC as appropriate.

Procedure Completion.
  Let $\overline{D}$ be temporary domains, initially D.
  Let TC be the set of constraints in CC.
  For all i add to TC constraints that exclude values of y that violate DC when ft= $\overline{ft}$.
  Apply an exact or heuristic algorithm to solve the constraints in TC using domains $\overline{D}$.
  If each |$\overline{D}_j$| = 1 then let D= $\overline{D}$.

Procedure Branching.
  Variable selection: Use guidelines in B to choose a branching variable st_j and to define subsets $D_j^1,\ldots,D_j^k$ of D_j.
  For l= 1,…,k:
    Let $D_j = D_j^l$ and add (CC,SC,DC,D) to A.
```

5. Conclusions

Our work focuses on solving the constraints in the logic way. In the discrete-variable logic, any constraint over finite domains in RCSP can be represented as a finite set of multivalent clauses. Using the discrete-variable formula to describe constraints can heighten the modeling capabilities of CP. Based on this model, we designed a branch and bound algorithm in the form of constraint-based search in the logic framework. In sum, logic-based modeling not only heightens the problem modeling capability but also exploits the problem solving method. Scheduling is really a process of getting the constraints right. However, designing a constraint model that can be used to solve real-life large-scale problems is also the biggest challenge of current Constraint Programming.

Acknowledgments

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] Barták, R.: On-line Guide to Constraint Programming. Charles University, Prague, http://kti.mff.cuni.cz/~bartak/constraints/.

[2] Wallace, M., Applying Constraints for Scheduling, in Constraint Programming[J], Mayoh B. and Penjaak J.(eds.), NATO ASI Series, Springer Verlag, 1994.

[3] Tsang, E. P. K, Foundations of Constraint Satisfaction[M]. San Diego, Calif.: Academic, 1993, pp.53-63.

[4] Juyang Zhang, Xin Li, Jigui Sun, Research On Constraint-based Scheduling and Its Implementation[A], Proceeding of CNCC'03[C], Beijing, P.R.China, The Tsinghua University Press, Nov., 2003, pp. 80-85.

[5] Jigui Sun, Juyang Zhang, A Generic Mechanism for Managing Resource Constraints in Preemptive and Non-Preemptive Scheduling[A], Processing of SCI'04 conference[C], Orlando, U.S.A., July, 2004.

[6] Kim Marriott, Peter J. Stuckey, Programming with Constraints: An introduction[M], The MIT Press, 1998, pp.133-134.

[7] Juyang Zhang, Jigui Sun, Qingyun Yang, Logic-based Constraint handling in Resource-constrained Scheduling Problem[A], Proceedings of International Conference on Computational Methods, Singapore, Spring Press, Dec., 2004.

Juyang Zhang received the M.S. degrees in Computer Software from Jilin University in 2003. His research interests include constraint programming, discrete optimization and intelligent scheduling..