대한산업공학회/한국경영과학회 2000 춘계공동학술대회 논문집. 2000년4월21-22일. 경남대학교
Session PA11.3

APPLICATION OF CONSTRAINT LOGIC PROGRAMMING TO JOB SEQUENCING

Jesuk Ko and Jaejung Ku

Department of Industrial and
Information Engineering, Kwangju University
592 Chinwol-dong, Nam-gu, Kwangju 502-703, Korea

Abstract In this paper, we show an application of constraint logic programming to the operation scheduling on machines in a job shop. Constraint logic programming is a new genre of programming technique combining the declarative aspect of logic programming with the efficiency of constraint manipulation and solving mechanisms. Due to the latter feature, combinatorial search problems like scheduling may be resolved efficiently. In this study, the jobs that consist of a set of related operations are supposed to be constrained by precedence and resource availability. We also explore how the constraint solving mechanisms can be defined over a scheduling domain. Thus the scheduling approach presented here has two benefits: the flexibility that can be expected from an artificial intelligence tool by simplifying greatly the problem; and the efficiency that stems from the capability of constraint logic programming to manipulate constraints to prune the search space in an a priori manner.

1. Introduction

Competitive manufacturing requires the efficient use of facilities to meet the cost and time requirements of customers; this is addressed by the scheduling of work orders within the manufacturing system. Many real-world manufacturing situations are particularly subject to difficult scheduling problems because they must deal with many constraints that must be satisfied for a successful solution. Most of them belong to the class of nondeterministic polynomial (NP)-complete problems [6, 7]. Also, as highly combinatorial search problems, they could be characteristic of the computational complexity. Much of the complexity comes from the need to attend to a large and diverse set of objectives, requirements and preferences that originate from many different sources in the plant-wide [21].

Logic programming, as exemplified by Prolog, provides a powerful language for a declarative formulation of these problems. Notwithstanding, due to the inefficiency of this search procedure based on the *generate-and-test* paradigm, logic programming languages have been used to solve toy problems so far. On the other hand constraint logic programming languages like CLP($\Re$) extend the scope of logic programming to numeric problem solving. So they provide strong mechanisms for constraints solving such as consistency techniques [17], basic arithmetic reasoning, and branch and bound for optimization [12]. In addition, they allow a programming style which often significantly reduces the computational complexity. These techniques have also been used in some systems to solve combinatorial problems [4, 5, 16]. The key idea behind constraint solving is to reduce the domains of variables and hence the search space to explore at each node of the tree. Thus they can be used to prune the search space in an a priori way rather than using them as tests leading to a generate-and-test procedure. This new paradigm exhibits a data-driven computation and can be characterized as *constrain-and-generate* [13, 14].

In this paper we show an application of such a constraint logic programming language, CLP($\Re$), to the job shop scheduling problem. The problem results from several factors: each job may require a different set of operations; the job has precedence; and there are resource constraints on these operations. The rest of paper is organized as follows. Section 2 presents a general description of a job shop scheduling problem. Section 3 describes a short overview of constraint logic programming languages. We then present the job shop scheduling problem which is solved in a constrain-and-generate manner. Also, we describe in detail how the problem can be stated and how constraint solving is performed. We report, in section 5, the computational results and their discussion. Section 6 contains the conclusions.

2. The Scheduling Problem

Consider the following scheduling problem. There are m machines and a set of n operations, where each operation is to be run on the specified machine. The executions of the operations are constrained by precedence constraints which are described by a directed acyclic graph $G = (O, P)$, referred to as the operation graph, where the set of vertices O corresponds to the set of operations and the set of arcs P to the precedence constraints. The operation graph is a weighted graph with vertices weighted by operation times op_i, $i \in O$, and arcs weighted by duration of delays d_{ij}. For any pair of operations $i, j \in O$, if $(i, j) \in P$, then operation j can start execution only d_{ij} time units after the completion of operation i, i.e., $c_i + d_{ij} \leq a_j$, where c_i is the completion time of operation i, a_j is the starting time of operation j. Also, resource constraints must be considered with respect to machine availability. The problem is to find a feasible schedule which satisfies the precedence and resource constraints such that the makespan, i.e. the time taken to complete all operations, is minimized. This problem is known as a job shop scheduling problem which minimizes the makespan. It belongs to the class of NP-complete problems [6, 7, 24]. For this class of problems the complete combinatorial search becomes enormous as the number of variables increases; the optimal solution frequently may not be easily found within an acceptable time limit. To solve this problem there are some heuristics which permit a near optimal solution. Some examples of heuristics are genetic algorithms [2], machine learning [18] simulated annealing [15] and Tabu search [8, 9, 10].

3. Brief Overview of CLP

The history of constraint logic programming can be traced back to 1987 when Jaffar and Lassez [13] introduced the basics of the constraint logic programming scheme, called CLP(X). X has been instantiated with several so called computation domains, e.g. reals in CLP($\Re$), rationals in CLP(Q) and integers in CLP(Z). By definition, constraint logic programming is a generalization of logic programming where unification, the basic operation of logic programming, is replaced by the more general concept of constraint handling (or solving) over computation domain. The major function of constraint solving mechanisms is constraint manipulation and propagation in a constraint domain.

The underlying idea behind constraint logic programming is the use of some mathematical tools to solve numerical constraints and the use of consistency checking and constraint manipulation techniques to solve symbolic constraints. These techniques admit computation directly over constraint domains such as algebraic operations, including set intersection, conjunction of boolean expressions or multiplication of arithmetic expressions. These computation domains also have certain relations like set equality, equality between boolean expressions or equality, disequality and inequality between arithmetic expressions. In the following section we will illustrate, as an operational behavior of a CLP(X), two constraint domains: arithmetic and boolean. Linear arithmetic expression is one of the motivations behind the research in combining logic programming with constraints. Arithmetic expression consists of terms composed from numbers, variables and the usual arithmetic operators like $+$, $-$, $*$, $/$. An arithmetic constraint is an expression of the form $a\ \theta\ b$ where θ is one of the following predicates $\{>, \geq, =, \leq, <, \neq\}$. This computation domain has been applied to combinatorial problems [4, 5, 16]. In contrast, boolean terms are built from the truth values (false and true, represented sometimes also by 0 and 1), from variables and from logical connectives (e.g. $\vee$, $\oplus$, $\wedge$, *neg*). The only constraint between boolean terms is the equality ($=$). The most prominent applications of boolean constraints are in the area of circuit design [20] and in theorem proving in the domain of propositional calculus [1].

Several types of constraint logic programming languages are in use today. Jaffar et al. (1992) developed the CLP($\Re$) system, which was the first constraint logic programming language to introduce arithmetic constraints [14]. In the internal mechanism of CLP($\Re$) the decision procedure of constraint solver is only complete for linear arithmetic constraints. When a nonlinear constraint is encountered during computation, then it is suspended and kept in a delayed constraint set. It is possible that some constraint in the delayed constraint set, at each operational step, need no longer be delayed because of new information. In this case it should be moved from the delayed constraint set to the collected constraint set and the usual solvability check made. The underlying constraint solving algorithm employs both an extended simplex method for inequality constraints and a Gaussian elimination algorithm for equality constraints respectively.

There are other interesting constraint logic programming languages such as PROLOG-III [1], CHIP [3], and Trilogy [25]; however, these other languages do not deal with constraints over real numbers. Most of these programming languages are commercially available and run on conventional, general-purpose computers because constraint logic programming hardware is at its infant stage.

4. A Scheduling Example

In this section, we illustrate by means of an example how our approach works. Suppose there are four different parts corresponding to a job which consists of a number of operation to be scheduled in two machines, mc1 and mc2. We shall also suppose that the operations (OP) that need to be performed on the specified machine and the precedence in which they must be executed are as follows:

Job_1 : OP_1 before OP_2

Job_2 : OP_3 before OP_4 before OP_5

Job_3 : OP_6

Job_4 : OP_7 before OP_8

Assume further that the machines are initially idle and the setup times are included in the job operation time. The operation time for various operations on each machine is given below:

OP_#	1	2	3	4	5	6	7	8
MC_#	1	2	2	1	1	2	1	2
OP_time	4	2	3	3	2	3	2	1

Now we will see how to represent the scheduling problem. The above example can be encoded in CLP($\Re$). It has the following structure: `job(Op, Duration, Prev, Res)`, with the reading : `job` *i*-th `Op`, which has the previous operation `Prev` and the operation time `Duration`; the operation is performed at the resource (machine) `Res`.

```
schedule([
    job(start,0,[],[]),
    job(op1,4,[start],[mc1]),
    job(op2,2,[op1],[mc2]),
    job(op3,3,[start],[mc2]),
    job(op4,3,[op3],[mc1]),
    job(op5,2,[op4],[mc1]),
    job(op6,3,[start],[mc2]),
    job(op7,2,[start],[mc1]),
    job(op8,1,[op7],[mc2]),
    job(end,0,[op8,op6,op5,op2],[])]).
```

Here the time of operation `start` / `end` - which can be regarded as job load / unload - is assumed to be 0. Now we are in a position to formulate the job precedence constraints. We introduce variables Op_i for each operation i. Using CLP($\Re$), these constraints can be expressed by numerical constraints of the following type:

$$Op_{i-1} + Duration_i \leq Op_i$$

In this way, all of the precedences can be considered. If operation $i-1$ has no successive operation, $Duration_i$ is equal to 0. In CLP($\Re$), all the constraints defining the problem are computed and then feasible solutions satisfying the constraints are generated. Consequently, after implementation, we can generate the following 12 inequalities:

```
start + 4 ≤ op1
op1   + 2 ≤ op2
start + 3 ≤ op3
op3   + 3 ≤ op4
op4   + 2 ≤ op5
start + 3 ≤ op6
start + 2 ≤ op7
op7   + 1 ≤ op8
op8   + 0 ≤ end
op6   + 0 ≤ end
op5   + 0 ≤ end
op2   + 0 ≤ end
```

The effect of these constraints is twofold: to prune the search space early and to escape deep backtracking. In addition, these constraints should be propagated when we consider the resource constraints (in this case two machines) together with precedence constraints. One way to define this characterization of a constraint in CLP($\Re$) would be the following procedure:

```
resources(Jobs,ScheduleEnd):-
   machine(Jobs,[],Mch),
   find_mch(Mch,Jobs,ScheduleEnd).

exclusive([],_,_,_,_).
exclusive(J.Js,Op0,Dur0,J0,Jobs,
     ScheduleEnd):-
     .....
   Op0 + Dur1 ≤ Op1 ; Op1 + Dur0 ≤ Op0
   exclusive(Js,Op0,Dur0,J0,Jobs,
      ScheduleEnd).
```

In the above program, the predicate `resources` selects each machine and assigns each operation to the specified machine. After choosing each set of operations, the predicate `exclusive` puts down the constraints with regard to operations assigned to the same machine. For instance, from Machine 1 we obtain the following 12 inequalities:

```
op7  + 2 ≤ op5 ; op5  + 2 ≤ op7
op7  + 3 ≤ op4 ; op4  + 2 ≤ op7
               :
op5  + 4 ≤ op1 ; op1  + 2 ≤ op5
op4  + 4 ≤ op1 ; op1  + 3 ≤ op4
```

From Machine 2, we also obtain the following 12 inequalities:

```
op8  + 3 ≤ op6 ; op6  + 1 ≤ op8
op8  + 3 ≤ op3 ; op3  + 1 ≤ op8
               :
op6  + 2 ≤ op2 ; op2  + 3 ≤ op6
op3  + 2 ≤ op2 ; op2  + 3 ≤ op3
```

The above 36 constraints define all the constraints of the present problem. Note that in the actual CLP ($\Re$) program, these constraints are automatically generated at the same time, and they can be used to drastically shrink the search space before the schedule is generated.

5. Results and Discussion

This section presents a detailed discussion of the results obtained in selected scheduling problems. As the result of computer implementation, we find a first solution for the job shop scheduling problem with a makespan of 20 after 0.02 second, and we obtain a total of 144 feasible solutions. Among them, there are 16 optimal solutions which minimize the makespan. One of the optimal schedule with a minimum makespan of 11 is found after 0.14 second. All our execution times are given for an IBM Pentium 133. In this paper, one of the interesting findings is the flexibility of schedule; in other words, the ability of scheduling to generate both rough and tight schedules in the goal state. The former are feasible schedules that just satisfy all constraints, while the latter are better schedules that minimize the makespan as well. In CLP($\Re$), as discussed previously, by using numerical constraints, the search space can be pruned very early. Also adding more constraints to the goal state, the schedule domain can be further reduced.

6. Conclusions

In this paper, we have presented the application of CLP($\Re$) to a job shop scheduling problem. CLP($\Re$) permits a declarative representation of the problem to be executed in an efficient way. The efficiency comes from the capability to manipulate constraints in order to prune the search space ahead of the schedule generation. The example discussed above is much simpler than those encountered in real-world scheduling problems, and the range of operations employed here would have to be widely expanded if a realistic problem were tackled. In addition, owing to its complexity, the constraints to be satisfied should be increased. In this case, specific heuristics could be developed and added without any difficulty. The aim of this paper is to show how the scheduling problem can be defined in CLP($\Re$). We also explore how to represent the constraint solving mechanisms over a scheduling domain. Our current research efforts are focused on implementing this approach to real problem and on improving algorithm performance by better maintaining the constraint solving over a computational domain.

References

[1] Colmerauer, A., 1990, An introduction to prolog-III. *Communications of the ACM*, 33(7), 69-90.

[2] Davis, L., 1985, Job shop scheduling with genetic algorithms. In J. J. Grefenstette(Ed.), International Conference on Genetic Algorithms and Their Applications, Carnegie-Mellon University, Pittsburgh, PA, 136-140.

[3] Dincbas, M., Hentenryck, P. V., Simonis, H., Aggoun, A., Graft, T., and Berthier, T., 1988, The constraint logic programming language CHIP. In Proceedings on the International Conference on 5th Generation Computer Systems (FGCS-88), Tokyo, Japan, December.

[4] Dincbas, M., Simonis, H., and Hentenryck, P. V., 1990, Solving large combinatorial problem in logic programming. *Journal of Logic Programming*, 8(1-2), 75-93.

[5] Fox, M. S., Allen, B. P., Smith, S. F., and Strohm, G. A., 1983, ISIS: a constraint-directed reasoning approach to job shop scheduling. Technical Report CMU-RI-TR-83-8, Carnegie-Mellon University.

[6] Garey, M. R., and Johnson, D. S., 1978, Strong np-completeness results: motivation, example and implication. *Journal of ACM*, 25, 499-508.

[7] Garey, M. R., and Johnson, D. S., 1979, *Computers and intractability: a guide to the theory of NP-completeness*, W. H. Freeman, San Francisco.

[8] Glover, F., 1989, Tabu search - part I, *ORSA Journal of Computing*, 1(3), 190-206.

[9] Glover, F., 1990, Tabu search - part II, *ORSA Journal of Computing*, 2(1), 4-32.

[10] Glover, F., and Laguna, M., 1992, *Tabu search in modern heuristic techniques for combinatorial problems*, Blackwell Publishing.

[11] Haralick, R. M., and Elliot, G. L., 1980, Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence, 14, 263-313.

[12] Hentenryck, P. V., 1989, *Constraint satisfaction in logic programming*, MIT Press, London.

[13] Jaffar, J., and Lassez, J-L., 1987, Constraint logic programming. In Proceeding of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany, January, 111-119.

[14] Jaffar, J., Michaylov, S., Stuckey, P. J., and Yap, R. H. C., 1992, The CLP($\Re$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3), 339-395.

[15] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C., 1991, Optimization by simulated annealing: an experimental evaluation. *Journal of Operations Research*, 39, 378-406.

[16] Lauriere, J-L., 1978, A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1), 29-127.

[17] Mackworth, A. K., 1977, Consistency in networks of relations. *AI Journal*, 8(1), 99-118.

[18] Nakasuka, S., and Yoshida, T., 1992, Dynamic scheduling system utilising machine learning as a knowledge acquisition tool. *International Journal of Production Research*, 30(2), 411-431.

[19] Pinedo, M., 1995, *Scheduling: theory, algorithms and systems*, Prentice Hall, New Jersey.

[20] Simonis, H., and Dincbas, M., 1987, Using logic programming for fault diagnosis in digital circuits. In Proceedings German Workshop on Artificial Intelligence (GWAI-87), Geseke, Germany, 139-148.

[21] Smith, S. F., Fox, M. S., and Ow, P. S., 1986, Constructing and maintaining detailed production plans: investigations into the development of knowledge-based factory scheduling systems. *AI Magazine*, 7(4), 45-61.

[22] Sule, D. R., 1997, *Industrial scheduling*, PWS Publishing, Massachusetts.

[23] Sussman, G. J., and Steele, G. L., 1980, Constraints - a language for expressing almost-hierarchical descriptions. *AI Journal*, 14(1).

[24] Ullman, J. D., 1975, NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10, 384-393.

[25] Voda, P., 1988, The constraint language Trilogy: semantics and computations. Technical Report, Complete Logic Systems, North Vancouver, Canada