Scheduling Using Constraint Programming

16 June 1997

Henrik Thane
Mechatronics Laboratory
The Royal Institute of Technology
Sweden
henrikt@damek.kth.se

Mårten Larsson
Real-Time Laboratory
Mälardalen University
Sweden
mln@idt.mdh.se

Abstract

In this document we will give a brief introduction to scheduling of hard real-time systems using constraint programming. We will give a short introduction to constraint propagation and distribution, describe how time triggered real-time systems can be described and how they can be scheduled using constraint programming.

Key words: constraint programming, real-time systems, off-line scheduling, Oz.

1 Constraint Programming

The class of problems that can be solved by constraint programming ranges from puzzles to scheduling, to ware house allocation, configuration and placement. In this brief introduction to constraint programming we will look into how constraint programming can be used for off-line scheduling of hard real-time systems. Studies have shown that constraint programming can also be utilized for tackling multi-rate systems with relative timing constraints but we will not cover these issues in this paper [Schild97].

1.1 Propagation and distribution

The two elementary techniques of constraint programming are **constraint propagation** and **constraint distribution**. Constraint propagation is an efficient inference mechanism using concurrently working propagators that accumulates information in a constraint store. Constraint distribution splits a problem into complementary cases when constraint propagation cannot advance any further. By iterating propagation and distribution, propagation will eventually determine the solutions of a problem.

An exponential growth in the number sub problems can easily arise from constraint distribution. The potential combinatorial explosion can however be limited by combining strong propagation mechanisms with problem specific heuristics for choosing the steps of distribution.

1.1.1 Finite domains and constraints

Finite domains are finite sets of nonnegative integers. Constraints are formulated by means of predicate logic. For example,

$$X = 67, \quad X \in \{0,\ldots,9\}, \qquad X = Y, \quad X^2 - Y^2 = Z^2, \qquad X + Y + Z < U, \quad X + Y \neq 5Z.$$

A **basic constraint** can e.g., take the form of $X = n$, $X = Y$ or $X \in D$, where n is a non-negative integer and D a finite domain. All constraints that are basic are located in a **constraint store**, which cache the conjunction of basic constraints up to logical equivalence. For example, a conjunction could be:

$$X \in \{0,..,5\} \wedge Y = 8 \wedge Z \in \{0,..,5\}$$

The **propagators** impose nonbasic constraints like, X < Y or $X^2 - Y^2 = Z^2$. A propagator for a constraint C is a concurrent computational agent that tries to narrow the domains of the variables occurring in C.

Two propagators that share the same variable can communicate with each other via the constraint store. For example, suppose we have these two propagators:

X +Y = 9 2X + 4Y = 24

over a constraint store with the following basic constraints:

$X \in \{0,\ldots,9\}$ $Y \in \{0,\ldots,9\}$

With the current ranges of X and Y the first propagator cannot do anything, however the second can narrow the domains of both X and Y:

$X \in \{0,\ldots,8\}$ $Y \in \{2,\ldots,6\}$.

Now can the first propagator narrow the domain of X:

$X \in \{3,\ldots,7\}$ $Y \in \{2,\ldots,6\}$.

Laboring on we now see that the second propagator can narrow the domain of both X and Y:

$X \in \{4,\ldots,6\}$ $Y \in \{3,\ldots,4\}$.

Using the first propagator again narrows the domain of X further:

$X \in \{5,\ldots,6\}$ $Y \in \{3,\ldots,4\}$.

And finally we use the second propagator again and determine the values of X and Y:

$X = 6$ $Y = 3$.

1.2 Distribution and search trees

Constraint propagation is usually incomplete (for the sake of efficiency). Hence to obtain a solution for a set of constraints *S*, we have to choose a (not necessarily basic) constraint *C* and solve both $S \cup \{C\}$ and $S \cup \{\neg C\}$. We say that we have distributed *S* with *C* at the current choice point. The second alternative is solved if the first alternative leads to an inconsistent store – meaning it has failed.

To illustrate this we give an example. Suppose we have these constraints and propagators:

$X \in \{1,\ldots,6,8\}$ $Y \in \{1,\ldots,10\}$ $Z \in \{1,3,\ldots,10\}$ $X - Z = 3Y$.

Using the propagator X - Z = 3Y we can narrow the domains to a stable set

$X \in \{4,\ldots,6,8\}$ $Y \in \{1,\ldots,2\}$ $Z \in \{1,3,\ldots,5\}$

The space is now stable but neither failed nor solved. So to go on we continue with a first distribution step. We choose to distribute with the constraint X = 4. Figure 1 shows the resulting search tree.

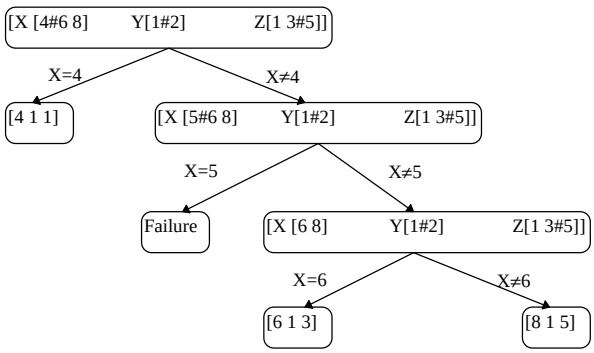


Figure 1 A search tree.

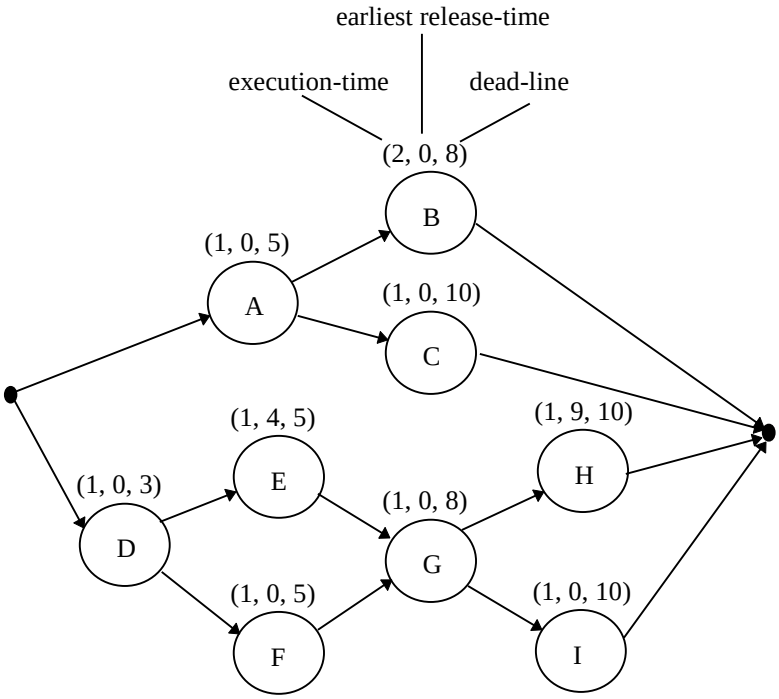
An alternative to the propagate and distribute method is a naive enumerate and test method, which would enumerate all triples (X,Y,Z) admitted by the initial domain constraints and test the constraints $X \neq 7$, $Z \neq 2$ and $X - Z = 3Y$ for each triple. If this strategy was to be employed there would be 8*10*10 possible candidates. This shows that constraint propagation can reduce the size of the search tree significantly.

The art of constraint programming consists of the discovery of a model describing the problem and devising a distribution strategy (heuristics) that yields a computationally feasible search tree

2 Scheduling using constraint programming

Constraint programming is well suited for solving scheduling problems of non-preemptive character. This is best shown in an example:

Consider the following precedence graph. The tasks are to be scheduled on the same node without preemption within a period of 10 time units.



First we can assert that the release time of a task is not greater than the deadline minus the execution time, and not less than zero

$$0 \leq release(A) \leq deadline(A) - execTime(A)$$

....

$$0 \leq release(I) \leq deadline(I) - execTime(I)$$

and then assert that all tasks are finished at the end of the period

$$deadline(A) \leq 10$$

...

$$deadline(I) \leq 10$$

and then the precedence relations are asserted

$$release(B) \geq deadline(A)$$

$$release(C) \geq deadline(A)$$

...

$$release(I) \geq deadline(G)$$

and finally it is asserted that there is no overlap between the execution of two different tasks

$$release(Task_i) + execTime(Task_i) \pounds release(Task_j)$$

$$or$$

$$release(Task_j) + execTime(Task_j) \pounds release(Task_i)$$

$$for\ all\ i \neq j$$

By then applying constraint propagation and distribution a solution is found. One possible solution is

$release(A) = 0$

$release(B) = 5$

$release(C) = 1$

$release(D) = 2$

$release(E) = 4$

$release(F) = 3$

$release(G) = 7$

$release(H) = 9$

$release(I) = 8.$

3 Conclusion

We have in this paper shown the basics of constraint programming and how constraint programming can be utilized for off-line scheduling. We did not address pre-emptive scheduling because this leads to very complicated constraints. As a comparison to regular off-line scheduling constraint distribution is no different. It is however in the use of constraint propagation that constraint programming excel. It can decrease the possible search tree significantly. Thus it would be a good idea to use constraint propagation as a preprocessing step before scheduling using more conventional techniques.

4 References

[Schild97] K Schild, J. Würtz. Scheduling of Time-Triggered Real-Time Systems. Draft, 1997. Programming Systems Lab. German Research Center for Artificial Intelligence (DFKI) and Universität des Saarlandes, Saarbrücken, Germany.

[Würtz96] J. Würtz. Constraint-Based Scheduling in Oz. 1996. Programming Systems Lab. German Research Center for Artificial Intelligence (DFKI) and Universität des Saarlandes, Saarbrücken, Germany.

[Smolka97] G. Smolka. Draft of the Finite Constraint Chapter of the Oz Primer, 25 March 1997. Programming Systems Lab. German Research Center for Artificial Intelligence (DFKI) and Universität des Saarlandes, Saarbrücken, Germany.