Why Is Scheduling Difficult?  
A CSP Perspective

Mark S. Fox
Department of Industrial Engineering  
University of Toronto  
4 Taddle Creek Road  
Toronto, Ontario M5S 1A4  
msf@ie.utoronto.ca

Norman Sadeh
Center for Integrated Manufacturing Decision Systems  
Carnegie Mellon University  
Pittsburgh, Pennsylvania 15213  
sadeh@cs.cmu.edu

DRAFT of June 3, 1994

Abstract
Interest in Constraint Satisfaction Problems (CSP) continues to grow, fueled by both their uniform problem representation, i.e., constraint graph, and conceptually clear problem solver, i.e., heuristically guided variable and value ordering. With the advent of interval constraints, e.g., temporal and spatial, and their associated consistency techniques, and the availability of Constraint Language Programming (CLP) it has become possible to explore complex problems such as planning and scheduling. This paper explores how a sequence of successively more complex scheduling problems can be modeled as a CSP, and the relevance of existing CSP problem solving heuristics. A number of problems arise with the CSP paradigm in modeling and solving scheduling problems: 1) Scheduling is an optimization problem in a very large combinatorial space. Therefore a good solution must be found as quickly as possible. 2) The existence of alternative process plans introduces disjunctive constraints in the constraint graph. 3) In most scheduling problems there exists a plethora of constraints. It is often the case that the problem is infeasible, requiring that one or more constraints be relaxed in order to find a solution. 4) Scheduling, and resource allocation problems in general, have constraint graphs in which many variables are tightly coupled (by capacity constraints) restricting the assignment of the same value to a single variable.

This research has been supported, in part, by the Defense Advance Projects Agency under contract F30602-88-C-0001, and in part, by grants from Digital Equipment Corporation and the McDonnell Douglas Corporation.
1. Introduction

Interest in Constraint Satisfaction Problems (CSP) continues to grow, fueled by at least three factors. First, the CSP model of problem solving is conceptually simple. There exists a single uniform problem representation composed of a constraint graph, and with this model, problem solving is reduced to constraint graph processing, and heuristically guided variable and value ordering [Mackworth 87]. The second factor was the advent of interval constraints [Davis 87], e.g, temporal [Allen 83] and spatial [Eastman 72], and their associated consistency techniques. This has made it possible to model complex problems such as factory scheduling and facility layout. The third factor was the creation of of Constraint Language Programming (CLP) [Cohen 90]. CLPs provide an environment in which the user simply models the problem as a system of constraints and lets the language determine how to solve it; methods are drawn from both Artificial Intelligence and Operations Research. The combination of these factors has made it possible to apply CSP methods to a larger set of realistic applications.

This paper explores how a sequence of successively more complex scheduling problems can be modeled as a CSP, and examines the relevance of existing CSP problem solving heuristics. Since the 1960s, the planning problem has captured the interest of many AI researchers. Planning selects and sequences activities such that they achieve one or more goals and satisfy a set of domain constraints. For the most part, planning research has focused on finding a feasible chain of actions that accomplish one or more goals. It was in the early 1980s that scheduling came under serious scrutiny, and more recently has garnered the attention of a significant minority of AI researchers, primarily in the domains of manufacturing and space. Scheduling selects among alternative plans, and assigns resources and times for each activity so that they obey the temporal restrictions of activities and the capacity limitations of a set of shared resources. Therefore scheduling is an optimization task where limited resources are allocated over time amongst both parallel and sequential activities. Both problems have been proven to be NP-Hard [Chapman 87, Garey & Johnson 79]. The recency of AI's focus on scheduling is somewhat odd, given that one of the earlier papers on planning explicitly pointed out the problem of allocating resources over time [Simon 72]. The sterile world of blocks never forced the issues that arise in scheduling; it took a return to the "real world" for these issues to reappear.

In earlier work, we have shown that one can view scheduling as a constrained
optimization problem [Fox 83, Fox 86]. This work solved the scheduling problem by using constraints to direct search in the problem space [Fox & Smith 84, Ow & Smith 88, Fox 90]. For example, precedence constraints can be used as operators to generate subsequent operations, and due date constraints can be used as part of the evaluation function to evaluate a state which depicts a partial schedule. The approach was synthetic in that it incrementally constructed a subset of partial schedules until one was found to be acceptable. Recently, a reductionist approach to scheduling, based on Constraint Satisfaction (CSP) techniques, has been explored. Techniques for constructing satisficing schedules [Elleby et al. 88, Keng et al. 88, Keng & Yun 89], and optimizing schedules [Sadeh & Fox 88, Sadeh & Fox 89, Fox et al. 89] have been demonstrated.

Viewing the scheduling problem from a CSP perspective can be useful. CSP is one of the few areas of AI where significant amounts of problem classification and complexity analysis has co-occurred [Mackworth 77, Haralick & Elliott 80, Freuder 82, Nudel 83, Purdom 83, Davis 87, Dechter & Pearl 87, Nadel 89]. Consequently, by reducing the scheduling problem to CSP, we can apply these results. On the other hand, as will be shown later, the scheduling problem extends beyond the current capabilities of CSP, providing for the carry over of methods from the scheduling domain.

In the following the complexity of the scheduling problem is explored through a series of factory scheduling problems. For each problem, an equivalent CSP formulation is provided. Finally, the relevance of CSP techniques is analyzed and approaches to solving them are briefly described.

2. Constraint Satisfaction Perspective of Scheduling

A constraint satisfaction problem [mackworth.89] is defined by a set of variables $V = \{v_1, v_2, \ldots, v_m\}$, each having a corresponding domain $D = \{d_1, d_2, \ldots, d_n\}$, and a set of constraints $C = \{c_1, c_2, \ldots, c_p\}$. A variable’s domain $d_i$ can be infinite, for example in the temporal domain, but is usually discrete and small. A constraint $c_j$ is a k-tuple that specifies a consistent assignment to the k variables that it constrains, i.e., $c_j \subseteq d_1 \times d_2 \times \cdots \times d_k$. Finding a solution to a CSP consists of finding a value assignment for each variable such that they satisfy all constraints.

The basic process of solving a CSP is comprised of the following steps:

1. select a variable for instantiation,
2. select a value to assign to the variable, and then
3. determine whether the assignment is consistent with all relevant

---

3This view is not unique; Operations Research has also formulated scheduling as an optimization problem and has applied various mathematical programming techniques, such as integer programming. See [Fox 91] for a comparison of AI and OR approaches.
constraints. If not, then backtrack, otherwise iterate.

The complexity of this search process can be reduced by a number of means. One method is to heuristically prioritize the order in which variables are selected. Another method is to heuristically prioritize the order in which values are selected for a particular variable. A third method, from which much of the power of CSP is derived, is called Arc-consistency. Arc-consistency pre-conditions a constraint network by selecting a single variable and checking to see if each value in its domain satisfies the relevant constraints. If a value is inconsistent, i.e., does not satisfy all the relevant constraints, then it is removed from the domain of the variable. Arc-consistency, in some cases, can greatly reduce the search process by reducing the size of the domains of many variables in the constraint graph.

CSP differs from conventional Heuristic Search practice in that it models a problem as a network of constraints and uses those constraints to reduce the domains of variables. CSP is a form of heuristic search; each time a variable binding is made, using variable and value ordering heuristics, the impact of that decision is propagated throughout the network via arc-consistency techniques.

In the reminder of this section a series of factory scheduling problems are described and their CSP analog is formulated.

3. Single Resource Scheduling

The simplest factory that one could imagine scheduling contains a single machine and produces a single product that requires a single operation. The scheduling goal is to assign each order for a product to an available time slot on the machine.

This can be stated more generally as a resource allocation problem where a single, indivisible resource, is to be allocated over time to \( n \) activities, but at any time to at most one activity. The activities are unrelated (i.e., no precedence relations among them) and are of equal duration and analyzed.

From a CSP perspective, this is equivalent to what we call the \textbf{NxM-Castle Problem}. Given an \( n \times m \) chess board, the problem is to place \( n \) castles such that they do not interfere according to chess rules. Unlike the N-Queens problem, more than one castle may occupy the same diagonal, but not the same row and column. Each castle, which occupies a separate row, corresponds to a separate activity and each column of the chess board corresponds to a unique, equal duration time slot that the activity can use the single resource.

More formally, given \( n \) activities \( A_i \) with domains, \( A_i \in \{1, \ldots , m\} \), assign a value to each subject to the following constraints:

\[
\forall ij [(i \neq j) \supset (A_i \neq A_j)] \text{No two distinct activities may occupy the same column.}
\]
An interesting feature of this problem is that the constraint graph, whose nodes are activities, is completely connected by inequality constraints that assure that no activities occupy the same column/time slot (figure 3-2). As long as the number of columns is greater than or equal to the number of castles, \( m \geq n \), then the problem is simpler than the N-Queens, and can be solved in linear time, by simply assigning each queen to the column equal to its row number.

With \( m=n \), there exist \( n! \) solutions, all equally good. But in scheduling, with \( m>n \), not all solutions are equally good. Within the Operations Research literature,
makespan how been used to measure of how long it takes to complete all of the jobs. The smaller the makespan, the better the machine is utilized, which is generally believed to be good. Consequently, scheduling is an optimization problem.

4. Scheduling with Due Dates

To bring some realism to the scheduling problem, we impose the constraint that each activity \( A_i \) must be completed before a "due date" \( d_i \). Each activity's due date is independent of the the due dates of other activities.\(^4\) This is the same as "mutilating" the NxM-Castle chess board by removing squares at the end of each row. Assuming that each column is numbered from 1 to \( m \), then if an activity \( A_i \) is due on date 5, then squares 6 through \( m \) in the activity's corresponding row \( i \) are unavailable for placing the castle.

![Figure 4-1: Mutilated NxM-Castle Problem](image)

More formally, given \( n \) activities \( A_i \) with domains \( A_i \in \{1, \ldots, m\} \), and due dates \( dd_i \). Assign a value to each subject to the following constraints:

\(^4\)Depending on how close the due dates are to each other, there may not exist sufficient time slots for all activities to be performed on or before their due dates.
\(\forall ij \ [i \neq j \supset (A_i \neq A_j)]\)

No two distinct activities may occupy the same column.

\(\forall i \ [A_i \leq dd_i]\)

An activity must end on or before its due date.

The additional due date constraints serve to reduce the domain of each activity variable prior to any search being performed. This problem can be solved in linear time, if a solution exists, by prioritizing activities by due date, and assigning the activity with the earliest due date to its due date or earlier.

In the manufacturing domain, there exists a preference for jobs to be scheduled as close to their due date as possible. Customers are given a due date and plan their operations around it. If the product arrives too early, they are not ready to use it and do not wish to pay for it. If the product arrives too late then their plans are no longer valid. **Earliness** is the measure of how early a job is completed before its due date. **Tardiness** is a measure of how late a job is completed after its due date. Scheduling with due dates is generally an optimization problem where both earliness and lateness are to be minimized. Scheduling activities in **shortest processing time** order has been proven to minimize tardiness [Baker 74].

5. Activities With Precedence

Consider the same factory, but now each product is produced according to a process plan. A process plan defines a sequence of operations (or activities) that must be performed in the order specified.

More generally, the single resource scheduling problem is now further complicated by imposing precedence among activities. That is, for each activity, there may be zero or more activities that must be performed before it.

Formally, the Mutilated NxM-Castle problem with precedence is defined as given \(n\) activities \(A_i\):

- with domains \(A_i \in \{1, \ldots, m\}\),
- due dates \(dd_i \in \{1, \ldots, m\}\), and
- a precedence matrix \(P_{ij}\), where \(P_{ij} = 1\) if \(A_i\) must precede \(A_j\), assign a value to each subject to the following constraints:
Figure 5-1: Mutilated NxM-Castle with Precedence

\[ \forall ij \, (i \neq j) \supset (A_i \neq A_j) \]
No two distinct activities may occupy the same column.

\[ \forall i \, (A_i \leq dd_i) \]
An activity must end on or before its due date.

\[ \forall ij \, (P_{ij} = 1) \supset (A_i < A_j) \]
Activities with precedence must be sequenced.

Initially, the constraint graph had all of the activity nodes connected together with inequalities, denoting that no castle/activity may occupy the same position. Precedence adds another layer of inter-activity constraints, as denoted by the \( P_{ij} \) precedence matrix. This problem can be solved in polynomial time (problem SS1 in [Garey & Johnson 79]).

6. Multiple Alternative Resources

To make the factory a little more realistic, more resources can be added. Now, each activity \( A_i \) may choose one of \( r \) resources to use, thus increasing the complexity of the task.

From a CSP perspective, the NxM-Castle problem can be further refined by extending the \( n \times m \) chess board into a third dimension, each plane representing an alternative resource (figure 6-1). No two castles may occupy the same column.
Figure 6-1: 3D Mutilated NxM-Castle with Precedence Problem

within the same plane.

More formally, given $n$ activities $A_i$ and $r$ alternative resources to choose from, with:

- activities having domains $A_i \in \{<T_i,R_i> | 1 \leq T_i \leq m, 1 \leq R_i \leq r\}$ where $T_i$ is the time (column position) of the activity and $R_i$ is the resource (plane) selected,
- due dates $dd_i \in \{1, \ldots, m\}$, and
- a precedence matrix $P_{ij}$ where $P_{ij} = 1$ if $A_i$ precedes $A_j$,

assign a value to each subject to the following constraints:

$$\forall ij \left[ ((i \neq j) \land (R_i = R_j)) \Rightarrow (T_i \neq T_j) \right]$$
No two distinct activities using the same resource may occupy the same time slot.

$$\forall i \left[ T_i \leq dd_i \right]$$
An activity must end on or before its due date.

$$\forall ij \left[ (P_{ij} = 1) \Rightarrow (T_i < T_j) \right]$$
If precedence exists among activities $i$ and $j$, then activity $i$ must be assigned a position before activity $j$. 
Nodes in the constraint graph are now 2-tuples; significantly enlarging the space of solutions. The choice of a time indirectly constrains the choice of a resource, and vice versa. The complexity added by the alternative resources makes this scheduling problem an NP-Complete problem in its general form. However, for precedence matrices that correspond to "in-tree" process plans, i.e., no activity has more than one successor, the problem can still be solved in polynomial time. "In-tree" process plans are assembly-type process plans (problem S11 in [Garey & Johnson 79]).

7. Interfering Activities

Activities may interfere if they are close to each other in time. Consider an assembly line, a type of flow shop, where a small number of product types are sequenced to be released to the line. Each type of product has a different set of component parts, or "options". If the assembly line can only handle one part with option $i$ out of every $b_i$ parts, the line should be balanced so that no two products of the same type be within $b_i$ positions of each other. Otherwise the line would have to be halted so that more time is available to complete the activities.

This is equivalent to an NxM-Castle problem, where castles can be of different colors, and the lines of interference are defined as regions. That is, there cannot be castles of the same color within a region centered at a castle, in the same plane.

Formally, given $n$ activities $A_i$ and $r$ alternative resources to choose from, with:

- activities having domains $A_i \in \{<T_i,R_i> | 1 \leq T_i \leq m, 1 \leq R_i \leq r\}$ where $T_i$ is the time or column position of the activity and $R_i$ is the resource or plane selected,
- due dates $dd_i \in \{1, \ldots, m\}$,
- a precedence matrix $P_{ij}$ where $P_{ij} = 1$ if $A_i$ precedes $A_j$,
- an activity type $AT_i \in \{1, \ldots, o\}$, that is, it is one of $o$ option types, and
- a neighborhood around a castle of type $t$ of size $B(t) \geq 0$,

assign a value to each subject to the following constraints:
\forall ij [(AT_i=AT_j) \land (R_i=R_j)] \supset
\left( |T_i-T_j| > B(AT_i) \right) \\
If activities \( i \) and \( j \) are of the same type and use the same resource, the "distance" between them should be greater than specified by its type.

\[ \forall i \left[ T_i \leq dd_i \right] \]
An activity must end on or before its due date.

\[ \forall ij [(P_{ij}=1) \supset (T_i < T_j)] \]
If precedence exists among activities \( i \) and \( j \), then activity \( i \) must be assigned a position before activity \( j \).

The delineation of regions around types of activities complicates the assignment problem. Poor initial assignments of activities may not leave open regions large enough for subsequent activities of the same type. Therefore assigning activities with the largest regions first would appear to be a good idea, and the smaller activities could be inserted among them.

Encouraging results in solving this problem using the Constraint Logic Programming language CHIP has been demonstrated [Dincbas et al. 88].

8. Non-Uniform Durations

The concept of interference among activities can be generalized further. Factories produce more than one product. Each product may have a different sequence of activities, the number of activities in a sequence may vary, and the duration of each activity may vary. It is the last point that complicates the scheduling problem further.

To accommodate this last point, the NxM-Castle problem is modified so that a castle may occupy more than one square, and the number of squares each castle occupies may differ.

Formally, given \( n \) activities \( A_i \) and \( r \) alternative resources to choose from, with
- activities having domains \( A_i \in \{<T_i,R_i> \mid 1 \leq T_i \leq m, 1 \leq R_i \leq r\} \) where \( T_i \) is the time or column position of the activity and \( R_i \) is the resource or plane selected,
- due dates \( dd_i \in \{1, \ldots, m\} \),
- a precedence matrix \( P_{ij} \) where \( P_{ij}=1 \) if \( A_j \) precedes \( A_i \),
- the number of squares occupied by the \( i \)th activity is defined by \( S_i \),
assign a value to each subject to the following constraints:

\[ \forall ij \left[(R_i = R_j) \supset ((T_j + S_{i-1} < T_i) \lor (T_i > (T_j + S_{i-1})))\right] \]

If two activities use the same resource, then they must not overlap in time.

\[ \forall i \left[(T_i + S_{i-1}) \leq dd_i\right] \]
An activity must end on or before its due date.

\[ \forall ij \left[(P_{ij} = 1) \supset ((T_i + S_{i-1} < T_j))\right] \]
If the \(i^{th}\) activity must precede the \(j^{th}\) activity, then the \(i^{th}\) activity must end before the \(j^{th}\) begins.

With non-uniform durations, choices of where to place an activity can have an enormous impact. If assignments do not leave gaps large enough for subsequent assignments, then significant amounts of backtracking can arise. One would think then that you would assign activities with the largest durations first.

9. Multiple Resources

It is usually the case that activities require more than one resource, thereby increasing the possibility of interference with other activities’ resource requirements.

Formally, given \(n\) activities \(A_i\) and \(m\) alternative resources to choose from, with

- activities having domains \(A_i \in \{<T_i,R_i> | 1 \leq T_i \leq m, R_i \in Res_i\}\) where \(T_i\) is the time or column starting position of the activity, \(R_i\) is the set of resources selected form \(R_i\) which defines the alternative sets of resources that activity \(i\) may use,
- due dates \(dd_i \in \{1, \ldots ,m\}\),
- a precedence matrix \(P_{ij}\) where \(P_{ij} = 1\) if \(A_i\) precedes \(A_j\),
- the number of squares occupied by the \(i^{th}\) activity is defined by \(S_i\),

assign a value to each activity subject to the following constraints:
∀i \[ (T_i + S_i - 1) \leq dd_i \]
An activity must end on or before its due date.

∀ij \[ (P_{ij} = 1) \Rightarrow ((T_i + S_i - 1) < T_j) \]
If the \(i^{th}\) activity must precede the \(j^{th}\) activity, then the \(i^{th}\) activity must end before the \(j^{th}\) begins.

∀ij \[ ((R_i \cap R_j) \neq \emptyset) \Rightarrow ((T_i + S_i - 1) < T_j) \lor (T_j > (T_i + S_i - 1)) \]
If two activities use the same resource, then they must not overlap in time.

With activities both restricted in the resources they may use and requiring more than one resource at a time, the degree of interference among activities continues to increase; the probability of an activity interfering with another is proportional to the number of resources it requires.

In the Operations Research literature, heuristic approaches to solving this problem have centered around scheduling bottlenecks first [goldratt.86, Adams et al. 88].

10. Variable Durations

The sequencing of activities at a specific resource may result in an expansion or contraction in the amount of time an operation will use one or more resources. For example, in the factory scheduling domain, an activity that uses the same setups, i.e., tooling, as the previous activity will not need as much set up time, thereby reducing the overall duration of the activity.

From a modelling perspective, context dependent operation durations can be viewed as a set of alternative operations, with different durations, and constraints that limit their selection based on the operation that precedes it at the same resource. That is, if the prior abutting operation is in the same family, then the operation with the reduced duration can be selected.

Formally, given \(n\) activities \(A_i\) and \(m\) alternative resources to choose from, with

- activities having domains \(A_i \in \{<T_i,R_i,V_i> | 1 \leq T_i \leq m, R_i \in Res_i\}\) where \(T_i\) is the time or column position of the activity, \(R_i\) is the set of resources selected form \(R_i\) which defines the alternative sets of resources that activity \(i\) may use, and \(V_i\) is the version of the activity whose choice is constrained by the activity that precedes it,

- \(Version_{ij}\) specifies the version of activity \(j\) if preceded by activity \(i\),
• due dates \( dd_i \in \{1, \ldots, m\} \),
• a precedence matrix \( P_{ij} \) where \( P_{ij} = 1 \) if \( A_i \) precedes \( A_j \),
• the number of squares occupied by the \( i^{th} \) activity is defined by \( S_{iv} \) (note that the amount depends on the version of the activity chosen),

assign a value to each activity subject to the following constraints:

\[
\forall i \left[ (T_i + S_{iv} - 1) \leq dd_i \right] \\
\text{An activity must end on or before its due date.}
\]

\[
\forall ij \left[ (P_{ij} = 1) \Rightarrow ((T_i + S_{iv} - 1) < T_j) \right] \\
\text{If the \( i^{th} \) activity must precede the \( j^{th} \) activity, then the \( i^{th} \) activity must end before the \( j^{th} \) begins.}
\]

\[
\forall ij \left[ ((R_i \cap R_j) \neq \emptyset) \Rightarrow \\
((T_i + S_{iv} - 1) < T_j) \lor (T_j > (T_i + S_{iv} - 1))) \right] \\
\text{If two activities use the same resource, then they must not overlap in time.}
\]

\[
\forall j \exists i \left[ \text{Directly-Precedes}(ij) \Rightarrow \\
(V_j = \text{Version}_{ij}) \right] \\
\text{If there exists an activity \( i \) that directly precedes activity \( j \), where Directly-Precedes is a predicate, then the activity’s version is specified by the matrix Version.}
\]

With durations being variable, it becomes more difficult to predict the impact of an assignment on subsequent assignments.

11. Continuous Time

Schedules are constructed to span a \textit{temporal horizon}, that is, detailed schedules are produced over some time interval. The length of the horizon depends upon the lead time with which resources, to be used in the production of the order, have to be planned and sourced. In some cases it may be weeks, and in other cases it may be years. Over the temporal horizon, schedules must describe activities to a particular \textit{temporal granularity}, perhaps to a day, shift, hour or minute. The temporal granularity of a schedule depends upon the duration of the activities, the length horizon, and the degree of uncertainty in the environment; that is, to what extent schedules can be followed due to stochastic events such a resource unavailability. Depending on the granularity the number of start times for each activity to choose from can be large. Short temporal horizons, tend to be scheduled in finer detail.
than those of greater spans.

Consider the simple scheduling problem with uniform durations and alternative resources. An upperbound on the number of solutions to this problem for the case of 100 activities, with 100 time slots, each having the choice of one of 100 resources is determined as follows. By restricting each activity/castle to a plane, each activity has at most $10^4$ positions to choose from. Given that there are $10^2$ activities, then the cross product of all these activities is $10^{400}$. For extended horizons or more precise granularities, the number of starting times is enormous.

**Figure 11-1:** Causal, Temporal, Resource Constraint Graph

At this point, viewing the problem from the NxM-Castle perspective becomes overly complex. It is time to explore a different representation that simplifies its conceptualization. Experience has shown that a change is required to an interval representation of time. Systems such as ISIS [Fox & Smith 84, Smith 83], Deviser [Vere 83] and OPIS [Ow & Smith 88] have used variants of Allen’s temporal relations [Allen 83] to represent scheduling intervals. Figure 11-1 illustrates such a representation where activities are nodes linked by causal, temporal interval and resource constraints. Temporal constraints constrain the start times of activities, and resource constraints limit the resources that an activity may use.

One approach to solving this problem has been to use a hierarchical organization of agents where higher level agents construct plans that are elaborated by lower level agents [Prosser 89].
12. Alternative Plans and Planning

A product may be fabricated and assembled in more than one way, reflecting the variety of processes and skills available. Scheduling in this scenario must consider alternative activities in addition to temporal and resource alternatives. For this scheduling problem, a process plan is defined by a graph of operations in which there exist alternative sub-graphs for fabricating and assembling a product. The complexity of scheduling is significantly increased with the introduction of alternative plans; we can no longer assume that each activity will be executed.

Figure 12-1: Causal, Temporal, Resource Constraint Graph

From a modelling perspective, precedence relations can be replaced by causal constraints that explicitly denote conjunctive and disjunctive precedence. The causal constraints denote disjunctive and conjunctive precedence among activities. The introduction of disjunctive causal constraints makes arc consistency a little more interesting!

The dynamic interleaving of planning and scheduling further complicates the problem. From a constraint graph perspective, planning introduces new nodes (i.e., activities) and constraints. The selection of one activity over another can have significant impact on the optimal utilization of resources over time. Relatively little work has been done on the optimal generation of plans from a scheduling perspective. Constrained Heuristic Search provides a model for this problem [Fox et al. 89]. Dynamic constraints provide another approach to including conditionally available activities [mital.89].

The Hubble Space Telescope Scheduler (HSTS) provides a powerful approach to the representation and reasoning about complex sequencing constraints in the planning and scheduling of telescope observations [Muscettola et al. 89].
13. Feasibility, Optimality and Relaxation

As mentioned earlier, the CSP perspective is that any variable assignment that satisfies the set of constraints is equally acceptable. This view treats scheduling as an allocation problem, where any allocation of resources over time that satisfy the constraints are equally acceptable. But this is not the case in the scheduling domain. Scheduling is really an optimization problem, the goal being to optimize objectives, such as:

- **Lateness**: Minimize the amount of time between when an activity is completed and its due date.

- **Flow Time**: Minimize the amount of time it takes for a product to complete its activities; that is, how much time the product spends being worked on in the factory.

- **Cost**: Minimize the amount of money spent on producing the product.

One AI approach to solving this problem, ISIS [Fox & Smith 84], has been to represent both objectives and constraints uniformly as constraints with associated priorities and utilities of satisfaction. These constraints are then used as heuristics in directing search in incrementally constructing schedules. OPIS [Ow & Smith 88] extends this approach to include two distinct scheduling perspectives, order-based and resource-based scheduling which it opportunistically switches between. An analysis of the problem’s constraints at each step of the search is used to choose among the perspectives. Other AI-based scheduling systems using similar approaches include RESS-1 [Liu 88], and SONIA [Collinot et al. 88].

The consequence of scheduling being an optimization problem is that the CSP definition is not sufficient to encompass the entire set of scheduling problems. Instead, a second class of problems, called Constrained Optimization Problems (COP) is defined similar to CSPs, but has, in addition, an **objective function**, that provides a numerical prioritization of proposed solutions. How the objective function is used during search is an interesting problem (see [Sadeh & Fox 88, Sadeh 91].

We have also taken for granted that a feasible solution exists to the aforementioned variations of the scheduling problem. This is not often the case in factory settings. Due to the costs involved, resources are not always available at levels that are sufficient to satisfy the temporal requirements imposed by due date and precedence constraints. Consequently, there may not exist a feasible solution.

In the factory scheduling domain it is not acceptable to just recognize that there does not exist a solution. Rather, as good a solution as possible must be found, even if a subset of the constraints are not satisfied. The question then is what subset of constraints are to be relaxed and how. Often, within a domain there is a clear weighting of constraints, and in some cases relaxations are specifiable. Some should not be relaxed, such as capacity constraints, but others can be relaxed, such as cost or due dates. This information may be utilized in the process of relaxation.
In NUDGE, knowledge of what and how to relax constraints was embedded in rules [Goldstein & Robert 77]. In ISIS and OPIS, relaxations and their utilities are defined explicitly as part of the constraint representation. A variety of techniques have been explored for deciding when and where to relax a constraint [Fox 90, Fox & Smith 84, Ow & Smith 88]. In CORTES [Sadeh & Fox 90a], constraints are represented in a similar manner, but algorithms are defined for the propagation of temporal, causal, and capacity preferences with the constraint network [Sadeh & Fox 88]. In Microboss, costs are propagated directly across constraints [Sadeh 91].

Alternatively, Operations Research would view a due date not as a constraint but as part of the objective function with the intent to minimize the difference between an activity's actual completion time and its due date. A fourth approach within the CSP framework randomly chooses constraints to remove from the problem [Freuder 89].

In order to address the issue of achieving feasibility via relaxation, the CSP constraint graph has to be extended to include information, such as the following:

- Relative weightings of constraints.
- Explicit specifications of relaxations of constraints.
- Utilities of relaxations.

Such representations have arisen in the constraint directed scheduling techniques discussed above, and in relaxation labelling techniques [Zucker 76].

14. The Job Shop Scheduling Problem
We have reached the point where we can now describe and represent the general job shop scheduling problem:

- The factory produces two or more different products, where similar products are grouped into families.
- Each product requires one or more operations to produce it, where the operations sequentially transform basic materials into the final product.
- The sequence of operations required to produce a product is defined to be its process plan. Process plans differ for each product.
- Each operation specifies one or more resources that are required during its performance. Resources are not longer functionally equivalent, each resource has limitations on the activities it can be used in.
- Durations for operations are specified apriori but may be reduced if orders for products in the same family follow each other on the same resource⁵.

An operation's duration is composed of a setup time and run time. The setup time is the amount of time required to prepare the resource for executing the operation. The piece time is the time to actually perform the execution of the operation for a single product. If two or more products of the same family are contiguous, then the second and subsequent setups are no longer required.
• Products are produced on demand, that is on the receipt of orders.
• There are multiple orders in production at any time. Contention usually exists for a subset of resources.
• Lead times for the delivery of orders can vary from zero days to multiples of the actual manufacturing lead time.
• A variety of objectives should be optimized, such as cost, tardiness, lateness, etc.

This problem is NP-hard and still awaits a good solution! It was the problem for which ISIS and OPIS were created to solve.

15. Relevance of CSP Search Techniques

Having described the scheduling problem from a CSP perspective, we examine the relevance of current CSP search techniques, in particular, variable and value ordering techniques. First, it is important to note that the CSP literature, by definition, has not addressed the issue of optimization, and has only recently considered the issue of feasibility and the need for relaxation that it entails\(^6\) Only recently, have the issues of Solving COPs been addressed [Fox et al. 89]. Consequently, our discussion only addresses the satisfaction aspect of the scheduling problem.

Significant reduction of the space of alternative schedules can be achieved by the preprocessing of the constraint network. For example, the satisfaction of temporal constraints, such as due dates and precedence, will reduce the start times of activities to a set of intervals. While the reduction can be substantial, it is often not enough.

The most widely accepted heuristic for selecting the next variable is to choose one with the smallest remaining domain [Bitner & Reingold 75]. It has been shown that this technique can greatly reduce search complexity [Purdom 83]. The question is whether it is sufficient for scheduling. In the scheduling domain, and for that matter in any domain in which resources are allocated over time, nodes in the constraint graph, i.e., activities, are a combination of temporal and resource variables. Simply looking at the number of starting times and/or resources remaining does not provide sufficient information to prioritize the variable. Instead, the capacity constraints introduce contention among the activities for the same resources. It is the degree of contention that needs to be captured in the variable ordering heuristic [Sadeh & Fox 88, Keng & Yun 89, Sadeh 89]. The greater the degree of contention that an activity faces for the resources it requires, the greater the probability that it will end up backtracking (see [Sadeh & Fox 90b] for a more detailed description).

---

\(^6\)We view existing constraint directed scheduling work, and vision-based relaxation labelling techniques as falling outside of conventional CSP literature.
For value selection, the concept of value goodness has been explored [Dechter & Pearl 87]. That is, choose a value that participates in the largest number of solutions. Since generating all possible solutions is not feasible, Dechter and Pearl proposed using a tree-like relaxation of the problem to approximate value goodness. For the scheduling problem we have found that it is the structure of the constraint graph, in particular its degree of completeness of capacity constraints, that make finding a solution difficult. Constructing a tree-like relaxation, such as required by ABT, tends to remove the constraints that make the problem difficult, reducing the "goodness" of the advice [Sadeh & Fox 90b]. A better way of computing value goodness in our class of problems is to again examine the degree of contention that exists for the value to be assigned.

**Figure 15-1:** Contention Graph

The role of contention in solving resource allocation problems is so important, that it becomes useful to introduce a *Contention Graph* (figure 15-1). A contention graph replaces capacity constraints by a node representing the resource whose capacity is constrained. At that node the aggregate demand, over time, from the activities is recorded and used in deciding both the variable and value to select.

### 16. General Scheduling Architecture

In the last five years, research in Artificial Intelligence constraint-directed approaches to scheduling have demonstrated a surprising degree of effectiveness and generality. The primary components of constraint-directed scheduling are a problem topology, textures and objectives. A problem topology is represented by a constraint graph. A problem topology may be altered by problem reformulations, such as abstraction and aggregation. Problem textures are fundamental measures of constraint graphs that indicate decision complexity, uncertainty and elasticity. Texture measures such as:
Value Contention: degree of to which more than one variable wish to have the same value.

Variable Reliance: degree to which a variable relies upon the assignment of a particular value.

Variable Looseness: size of range (conjunction of constraints).

Constraint Tightness: degree to which the constraint reduces the set of admissible solutions.

Constraint Importance: how important is it to satisfy the constraint.

are used to determine where in the constraint graph the next decision is to be made, i.e., variable and constraint selection. Problem objectives define what is to be optimized.

Two methods for assigning resources over time to each of the activities have been used. The generative method starts with an empty schedule and iteratively selects and assigns resources to each activity over time, and backtracks whenever a deadend (i.e., infeasibility) is reached. The repair method starts with a completed scheduled containing unsatisfied (i.e., broken) constraints and iteratively selects and reassigns resources to an activity over time that satisfy as many unsatisfied constraints as possible. Search in both cases ends whenever the best schedule is found that satisfies the constraints. The repair method can be augmented by using simulated annealing to find better schedules [zweben.92].

Based on these observations, we are developing a Constraint-directed scheduling shell that integrates the generative and repair approaches around a single constraint-graph representation of the problem [davis.fox.93]. The shell's architecture (Figure 16-1) provides for four modules, each further constraining the scheduling solution.

Specification: The specification module provides a user interface for the acquisition of the scheduling problem description. Information such as the tasks, resources, variables, constraints, etc. are acquired through one or more graphic perspectives of the problem (e.g., Gantt chart). In addition, the problem specification may be changed during the scheduling process. Information may be added, modified or retracted.

Generation: The generation module creates a seed solution which will be improved by the repair module. The intent is to generate as good a solution as possible in order to facilitate the repair process. A solution is generated by iteratively performing arc-consistency, texture measurement, then using the textures to heuristically select a variable and assign to it a value [Sadeh 91]. Before generation, the only variables that have been assigned values, are those specified by the user. The precision of the schedule will depend on an analysis of domain uncertainty. Therefore, the output of this module is a set of constraints that significantly
reduce the set of possible schedules [Fox & Smith 84, Fox & Sycara 90, prosser.90].

**Repair:** The repair module iteratively improves the seed solution provided by generation step. It uses simulated annealing to avoid local minima/maxima. The identification of repair points use the same textures that were used by the generation module. The repair module's output further constrains the set of possible schedules to be executed by the execution module.

**Execution:** The execution module takes as input a set of constraints that limit the flexibility of execution. For example, the constraints limit the resources to be used by an activity and the time period in which it can occur. The execution module may flexibly respond to dynamic events within these limits. The greater the domain uncertainty, the greater the flexibility will be inherent in the schedule. If an event occurs that can be dealt with within the constraints provided, then the event is signaled to the repair module for its handling.

**Figure 16-1:** Shell Architecture

17. Conclusion

Constraint Satisfaction research has made great strides in understanding the power of heuristics in solving large systems of constraints. But the continued focus on N-Queens and satisfiability problems has led CSP research to ignore additional problem characteristics and rich constraint structures that arise in specific classes problems. Our experience in the scheduling domain has demonstrated both the relevance and limitations of CSP techniques. What makes scheduling a difficult problem to solve from a CSP perspective is:

1. Scheduling is an optimization problem in a very large combinatorial space. Therefore a good solution must be found as quickly as possible.

2. The existence of alternative process plans introduces disjunctive constraints in the constraint graph.

3. In most scheduling problems there exists a plethora of constraints. It is often the case that the problem is infeasible, requiring that one or more constraints be relaxed in order to find a solution.

4. Scheduling, and resource allocation problems in general, have constraint graphs in which many variables are tightly coupled (by capacity constraints) restricting the assignment of the same value to a single variable. The larger the number of coupled variables, the more difficult the assignment problem. A contention graph is introduced to aid in providing better measures of the graph and to guide in variable and value selection.

5. Uncertainty affects the precision with which decisions can be made. Understanding the affect of uncertainty on precision is still poorly
understood.

References

[Adams et al. 88] Adams, J., Balas, E., and Zawack, D.
The Shifting Bottleneck Procedure for Job Shop Scheduling.

[Allen 83] Allen, J.F.
Maintaining Knowledge about Temporal Intervals.

[Baker 74] Baker, K.R.
*Introduction to Sequencing and Scheduling.*

[Bitner & Reingold 75]
Bitner, J.R., and Reingold, E.M.
Backtrack Programming Techniques.

[Chapman 87] Chapman, D.
Planning for Conjunctive Goals.

[Cohen 90] Cohen, J.
Constraint Language Programming.

[Collinot et al. 88] Collinot, A., Le Pape, C., and Pinoteau, G.
SONIA: A Knowledge-Based Scheduling Systems.

[Davis 87] Davis, E.
Constraint Propagation with Interval Labels.

[Dechter & Pearl 87]
Dechter, R. and Pearl, J.
Network-based Heuristics for Constraint-Satisfaction Problems.

[Dincbas et al. 88]
Dincbas, Simonis, and Van Hentenryck.
Solving the Car-Sequencing Problem in CLP.
[Eastman 72] Eastman, C.

Reactive Constraint-Based Job-Shop Scheduling.
Expert Systems and Intelligent Manufacturing.
In Oliff, M.D.,

[Fox 83] Fox, M.S.
Constraint-Directed Search: A Case Study of Job-Shop Scheduling.

[Fox 86] Fox, M.S.
Observations on the Role of Constraints in Problem Solving.

[Fox 90] Fox, M.S.
Constraint Guided Scheduling: A Short History of Scheduling Research at CMU.

[Fox 91] Fox, M.S.
Reflections on the Relationship of AI to OR.
Interfaces, 1991.
To Appear.

[Fox & Smith 84] Fox, M.S., and Smith, S.F.
ISIS: A Knowledge-Based System for Factory Scheduling.

[Fox & Sycara 90] Fox, M.S. and Sycara, K.
Overview of CORTES: A Constraint Based Approach to Production Planning, Scheduling and Control.

[Fox et al. 89] Fox, M.S., Sadeh, N., and Baykan, C.
Constrained Heuristic Search.
[Freuder 82] Freuder, E.C.
A Sufficient Condition for Backtrack-free Search.

[Freuder 89] Freuder, E.C.
Partial Constraint Satisfaction.


*NUDGE: A Knowledge-Based Scheduling Program.*
AI Memo 405.

[Haralick & Elliott 80] Haralick, R.M., and Elliott, G.L.
Increasing Tree Search Efficiency for Constraint Satisfaction Problems.

[Keng & Yun 89] Keng, N.P., and Yun, D.Y.Y.
A Planning/Scheduling Methodology for the Constrained Resource Problem.

[Keng et al. 88] Keng, N.P., Yun, D.Y.Y., and Rossi, M.
Interaction-sensitive planning system for job-shop scheduling.

[Liu 88] Liu, B.
Scheduling via Reinforcement.

[Mackworth 77] Mackworth, A.K.
Consistency in Networks of Relations.

[Mackworth 87] Mackworth, A.K.
Constraint Satisfaction.
*Encyclopedia of Artificial Intelligence.*
In Shapiro, S., J. Wiley & Son, 1987.
Generating Space Telescope Observation Schedules.  

[Nadel 89] Nadel, B.A.  
Constraint Satisfaction Algorithms.  
Computational Intelligence 5(4):188-244, 1989.

[Nudel 83] Nudel, B.A.  
Consistent Labelling Problems and their Algorithms: Expected-Complexities and Theory-Based Heuristics.  

[Ow & Smith 88] Ow, P-S., and Smith, S.F.  
Viewing Scheduling as an Opportunistic Problem-Solving Process.  

[Prosser 89] Prosser, P.  
A Reactive Scheduling Agent.  

[Purdom 83] Purdom, P.W., Jr.  
Search Rearrangement Backtracking and Polynomial Average Time.  

[Sadeh 89] Sadeh, N.  
Lookahead Techniques for Activity-based Job-shop Scheduling.  
Thesis Proposal.

[Sadeh 91] Sadeh, N.  
Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling.  

[Sadeh & Fox 88] Sadeh, N., and Fox, M.S.  

[Sadeh & Fox 89] Sadeh, N. and Fox, M.S.  
Focus of Attention in an Activity-based Scheduler.  
1989.
[Sadeh & Fox 90a]
Sadeh, N., and Fox, M.S.
Variable and Value Ordering Heuristics for Activity-Based Job-Shop Scheduling.

[Sadeh & Fox 90b]
Sadeh, N., and Fox, M.S.
*Generating Advice for Hard Constraint Satisfaction Problems: An Application to Job-Shop Scheduling*.
In Preparation.

[Simon 72] Simon, H.A.
On Reasoning About Actions.
*Representation and Meaning: Experiments with Information Processing Systems*.
Prentice-Hall, 1972, pages 414-430.

[Smith 83] Smith, S.F.
*Exploiting Temporal Knowledge to Organize Constraints*.

[Vere 83] Vere, S.
Planning in Time: Windows and Durations for Activities and Goals.

[Wilkins 88] Wilkins, D.E.
*Practical Planning*.

[Zucker 76] Zucker, S.W.
Relaxation Labelling and the Reduction of Local Ambiguities.
*Pattern Recognition and Artificial Intelligence*.
Table of Contents

1. Introduction 1
2. Constraint Satisfaction Perspective of Scheduling 2
3. Single Resource Scheduling 3
4. Scheduling with Due Dates 5
5. Activities With Precedence 6
6. Multiple Alternative Resources 7
7. Interfering Activities 9
8. Non-Uniform Durations 10
9. Multiple Resources 11
10. Variable Durations 12
11. Continuous Time 13
12. Alternative Plans and Planning 15
13. Feasibility, Optimality and Relaxation 16
14. The Job Shop Scheduling Problem 17
15. Relevance of CSP Search Techniques 18
16. General Scheduling Architecture 19
17. Conclusion 21
References 22
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>NxM-Castle Problem</td>
<td>4</td>
</tr>
<tr>
<td>3-2</td>
<td>Constraint Graph</td>
<td>4</td>
</tr>
<tr>
<td>4-1</td>
<td>Mutilated NxM-Castle Problem</td>
<td>5</td>
</tr>
<tr>
<td>5-1</td>
<td>Mutilated NxM-Castle with Precedence</td>
<td>7</td>
</tr>
<tr>
<td>6-1</td>
<td>3D Mutilated NxM-Castle with Precedence Problem</td>
<td>8</td>
</tr>
<tr>
<td>11-1</td>
<td>Causal, Temporal, Resource Constraint Graph</td>
<td>14</td>
</tr>
<tr>
<td>12-1</td>
<td>Causal, Temporal, Resource Constraint Graph</td>
<td>15</td>
</tr>
<tr>
<td>15-1</td>
<td>Contention Graph</td>
<td>19</td>
</tr>
<tr>
<td>16-1</td>
<td>Shell Architecture</td>
<td>21</td>
</tr>
</tbody>
</table>