An Architecture for Robust, Reactive Constraint-Based Scheduling

Aditya K. Ghose and Joseph G. Davis
Department of Business Systems
University of Wollongong, NSW 2522 Australia
{aditya, j.davis}@uow.edu.au

Abstract

This paper seeks to motivate and describe the RR-CONS architecture for robust, reactive constraint-based scheduling. Our work improves on existing approaches to constraint-based scheduling by providing a seamless integration of the following key features: robust handling of multimodal input specifications, incremental rescheduling, real-time scheduling/rescheduling and mixed-initiative reasoning.

1 Introduction

Scheduling refers to the problem of assigning resources to activities in some (ideally) optimal sequence. It is a significant, and ubiquitous, aspect of a wide variety of business and industrial processes. Thus, airline crew scheduling, hospital nurse rostering, job shop scheduling in manufacturing units, university class timetabling, and the assignment of aircraft to airport terminal gates are all manifestations of the same fundamental scheduling problem. It is possible to identify a set of common features for the scheduling task in each of these settings. The scheduling process must handle complex specifications of input knowledge and must attempt to satisfy a number of different, and often conflicting optimization criteria. The inputs to the scheduling process are prone to frequent change and the outputs of the scheduling process, i.e., the generated schedules, have to be revised to reflect these changes, often in real-time. The process of searching for the best schedule is computationally hard and user inputs can often become necessary at various points in the scheduling process in order to generate reasonable conclusions within an acceptable time-frame. Scheduling is often an exercise in distributed problem-solving, requiring the use of complex coordination and negotiation protocols involving site-specific scheduling processes spread out over several geographically dispersed sites. Scheduling problem-solvers are often part of large enterprise-wide information systems and must provide seamless interfaces to other components of such systems. Finally, scheduling systems are often required to play a role in organisational decision support and must support forecasting, hypothetical reasoning and what-if analysis functions.

Despite the growing maturity of scheduling tools, many existing scheduling systems have not lived up to expectations due to their inflexibility, inordinately high response time, and the inability of users to make up-to-the-minute updates to reflect changing conditions in order to generate new solutions dynamically and in real-time. This has created a situation in which several expensive scheduling systems find little or no use in the organizations that deploy them. This is partly a reflection of the limitations in traditional scheduling methods. Mathematical programming methods (linear, integer, mixed integer) from operations research, for instance, represent a mature toolkit of techniques which are severely limited in their expressive power. Thus, large classes of scheduling problems cannot be reduced to representations that linear programming systems take as input. As well, such systems cannot handle the kinds of defeasible multi-criteria optimization issues that need to be addressed for realistic problem domains, nor respond in real-time to changing input specifications. Similar arguments apply to a large class of other traditional scheduling methods.

The area of *intelligent scheduling* (Zweben and Fox 1994) seeks to address some of the deficiencies in traditional approaches by providing a richer representation for both the input knowledge, the system objectives, and optimization criteria. One approach within this area has viewed scheduling as a knowledge-based problem solving exercise (see for example Trzewik et al. 1994), with both rule-based production systems approaches as well as formal approaches to knowledge representation. An alternative approach, and one that is becoming increasingly popular and (arguably) more successful, views scheduling as a *constraint satisfaction problem (CSP)*. In a CSP, the problem is specified using a set of variables with pre-defined domains and a set of *constraints* on the values these variables may take. Much of the recent work on intelligent scheduling has focussed on graph-based constraint satisfaction and optimization techniques (Zweben et al. 1992, Sadeh 1991, Keng and Yun 1989) in which problems are represented as constraint graphs with variables as nodes (representing tasks and resources) and constraints as edges. In constructive approaches to solving such problems

(Fox 1987, Sadeh 1991, Keng and Yun 1989), starting with an empty schedule, values are assigned to variables by checking for consistency with prior assignments until a solution is reached. If the current set of assignments cannot lead to a solution, the algorithm backtracks and tries alternative assignments. In iterative approaches to solving such problems (Zweben et al. 1993, Minton et al. 1992), the algorithm starts with a complete assignment of values to all variables and incrementally alters these assignments until a consistent solution is reached. Scheduling systems such as ODO (Davis and Fox 1994) combine elements of both the constructive and iterative solution methods. An interesting set of hybrid approaches have emerged in the recent past that combine elements of both the knowledge-based (i.e., non-constraint based) approaches and constraint-based approaches to scheduling. These are based on the *constraint logic programming* paradigm (see Jaffar and Maher 1994, for a good survey) which combine the well-known capabilities of Horn clause programming with the representational flexibility and efficient query processing capabilities (using efficient solvers for special classes of constraints) of constraint-based representations. Several existing constraint logic programming languages, such as CLP(R) (Jaffar et al. 1992), Echidna (Havens et al. 1992) and Nicolog (Sidebottom and Havens 1995) have been successfully deployed in intelligent scheduling applications.

This paper seeks to motivate and describe the RR-CONS architecture for robust, reactive constraint-based scheduling. Our proposed approach to the scheduling problem builds on constraint-based scheduling techniques, but there are significant differences between our proposal and existing work. Our commitment to a constraint-based scheduling framework has been motivated by the two following observations. First, constraints provide an expressive, yet flexible, language for representing knowledge in a large class of domains. Scheduling problems, in particular, find natural expression in constraint-based languages. Second, constraints lend themselves to fast inferencing procedures. A variety of efficient constraint-solving techniques exists for a large class of constraint domains.

The rest of this paper is organized as follows. In the next section, we shall motivate the new architecture via a critique of the state-of-the-art in constraint-based intelligent scheduling. This focus notwithstanding, our arguments apply, to varying degrees, to all existing scheduling methods/systems. In section 4, we outline the RR-CONS architecture. In Section 5, we describe the *Specification Language* of the RR-CONS architecture in some detail.

2 A critique of existing techniques

The RR-CONS architecture was initially developed as part of an industry collaborative project (currently underway) to develop a scheduling system for the fabrication and delivery tasks of a major house-frame manufacturing unit. In the rest of this section, we shall use examples from this domain to illustrate some of our arguments against existing constraint-based scheduling systems.

- **Robust handling of input specifications:** Existing scheduling systems operate under strict assumptions regarding the consistency of the input specifications. Thus, a constraint-based scheduler is able to generate meaningful schedules only when provided with a solvable set of constraints as input. One method of accounting for this requirement is to perform manual pre-processing of the input specification to guarantee consistency. This, however, turns out to be impossible for industrial-scale applications with large input specifications. A more common approach is to manually adjust the input specification when the scheduling system flags an inconsistency (by deleting a subset of the input constraints, for instance). This too is a tedious, time-consuming process which must, in principle, consider every permutation of the input specification in order to generate all of its maximal consistent subsets. The scheduling system, based on the RR-CONS architecture, being developed for the house-frame manufacturing unit is intended to be operated by shop-floor personnel; it is perhaps unrealistic to expect such users to pre-process the inputs to guarantee consistency or to adjust them to remove constraints that might lead to inconsistency. The requirement for consistent inputs is thus a clear deterrent to scaling up scheduling systems to industrial-grade applications.
- **Multimodal representations of input specifications:** Most existing scheduling systems impose severe restrictions on how the input knowledge is represented. The input representations are typically *unimodal*, i.e., constraint-based scheduling systems accept constraint-based representations, rule-based scheduling systems accept production rules etc. However, the inputs to industry-grade scheduling applications tend to be complex and often do not find natural expression in any single knowledge representation scheme. Our client house-framing unit, for instance, currently uses a *scheduling assistant* (as opposed to a full-fledged scheduling system) for providing limited auto-

mated support to human schedulers for generating practical house-frame fabrication and delivery schedules. The scheduling assistant uses *assertional* (i.e., non-constraint-based) knowledge in the form of rules that enable it to generate an initial design specification for a house-frame based on the identity of the home-builder and the type of home to be constructed (under the assumption that each home builder has a limited repertoire of designs, one for each type). The scheduling assistant also uses *hard* (or inviolate) job-shop constraints (although these are currently procedurally encoded in the system). Human schedulers at the manufacturing plant constantly manipulate *soft* (or defeasible) constraints in the form of delivery deadlines (which may be violated), prioritizing them on the basis of a number of contingent factors such as the standing of the customer. Realistic scheduling knowledge thus comes in *multi-modal* representations.

- **Incremental rescheduling:** The inputs to the scheduling process are prone to constant change. Thus, human schedulers at the house-frame manufacturing plant must constantly deal with requests from customers to move up delivery deadlines as well as the possibility of machine breakdowns on the production line, inventory problems, or critical personnel absences. A crucial aspect of effective scheduling frameworks, and one that existing systems tend to ignore, is the capability of performing *incremental rescheduling*, i.e., generating revised schedules in the face of changing inputs by modifying existing schedules, as opposed to computing the revised schedule from scratch. Given the time-critical nature of the scheduling process, incremental rescheduling capability is often the crucial factor that determines whether a scheduling system scales up to an industrial-grade application.

- **Real-time scheduling/rescheduling:** As noted above, scheduling applications are typically time-critical, with schedules having to be generated in real-time. In the house-frame manufacturing unit's scheduling application, a rush order from a major customer requires the recomputation of fabrication schedules in a matter of minutes to accommodate the new request. When a builder reports on-site problems that necessitate delaying delivery even as a truck is being loaded with complete knock-down (CKD) house frame kits for a delivery route that involves multiple drop-off points, a new schedule needs to be computed under even more stringent time constraints, given that delaying the loading and departure of the truck would result in other hard delivery deadlines not being met. Most existing scheduling frameworks provide no support for real-time scheduling/rescheduling (although Zweben et al. 1990, is an exception), thus bringing into question their viability in realistic settings.

- **Mixed-initiative reasoning:** Given the complexity and inherent intractability of the scheduling problem, there are clear benefits in terms of both the speed-up of the scheduling process and the improvements in solution quality that would accrue if the scheduling system were to query the user at crucial decision points. This is in line with the philosophy, gaining increasing currency amongst artificial intelligence researchers, that the best solution to the inherent intractability of real-life inference is to make the process *mixed-initiative*. In mixed-initiative reasoning, the process of inference is driven (hence the notion of *initiative*) by both the user and the system; typically, this implies that the system seeks the users input at various points in the problem solving process. Similar considerations drive the focus, in decision support systems, to *informate* decision processes, as opposed to *automating* them. Existing scheduling systems tend to ignore the possibilities offered by mixed-initiative reasoning, resulting unnecessary backtracking at various levels of abstraction in the problem solving process, with concommitant losses in efficiency.

3 The RR-CONS scheduling architecture

There are five key elements of the RR-CONS architecture that distinguish it from other constraint-based scheduling architectures.

- It supports a contradiction-tolerant, mixed-mode input specification module. The module allows assertional, constraint-based and graphical representations. In each of these representations, it is possible to distinguish between the mutable and immutable components. While most existing scheduling frameworks support input specification in any one of these modes, we are not aware of any that support specifications in all three modes. The ability to distinguish between the mutable and immutable components (i.e., between the defaults and facts) is another novel feature that enables robust handling of complex inout specifications.

- It permits formal analysis and correctness proofs of scheduling systems using a uniform underlying semantics for the complex mixed-mode input specifications.

- It allows efficient processing of such complex inputs using a single scheduling engine. Specifically, we are able to translate input specifications in each of these three modes into a uniform constraint-based representation. It is then possible to use any standard library of constraint solvers (or a commercial product such as ILOG SOLVER) as the scheduling engine.

- The architecture supports real-time scheduling using anytime algorithms, described later in this section.

- The architecture support updates to scheduling problem specifications, coupled with procedures for incremental, dynamic rescheduling.

The RR-CONS architecture consists of three major modules:

- **The Input Specification Module:** The architecture supports multi-modal representations of input knowledge in the following forms:

  - *Assertional representations* which support the distinction between inviolate assertions and potentially defeasible ones.
  - *Constraint-based representations* which support the distinction between hard and soft constraints. Current scheduling systems rarely support an explicit notion of soft constraints, (Beck 1994) being a notable exception.
  - *Graphical representations:* Inputs may also be provided in the form of partially specified PERT graphs or Gantt charts. It is easy to see that these graphical tools provide a powerful medium for non-technical users to communicate scheduling constraints to the system.

  The first two components of the multi-modal input specifications are integrated into a formal *specification language* (based on the notion of *Non-monotonic Constraint Logic Programming*, Ghose 1997) to be described in the next section. The specification language supports formal notions of defeasible and non-defeasible logical assertions, as well as explicit notions of hard and soft constraints. For the last component, the module supports PERT graph and Gantt chart editors.

We make a fundamental distinction between *competence theories* and *performance theories* in the context of input representations. We commit to the formal notion of *default representations* (as in default logic, Reiter 1980, or hypothetical reasoning systems, Poole 1987) for defining a competence theory for the problem specification. Any claims of correctness of the various operations defined on the input representation would thus make appeal to the well-defined semantic underpinnings provided by a uniform translation of every component of the input specification into default representations. For the purposes of a performance theory, we utilize a uniform translation from the formal specification language as well as the graphical representations into constraint-based representations. The intent, thus, is to capitalize on efficient constraint-solving techniques for the purposes of problem solving.

- **Translation Module:** The task of the translation module is to map every component of a multi-modal input specification of a scheduling problem into an equivalent constraint-based representation. It consists of three distinct translators:

  - *SpecificationLanguage-Constraints Translator*
  - *PERTGraph-Constraints Translator*
  - *GanttChart-Constraints Translator*

  The SpecificationLanguage-Constraints Translator exploits a mapping from default theories to partial constraint satisfaction problems defined in (Ghose et al. 1998) while the PERT graph and Gantt chart translators affect the relatively straightforward mappings to constraint-based representations.

- **Revision Module:** A crucial component of a reactive scheduling architecture is its ability to incorporate changes to its problem specification. Following the well-established literature on belief change (Alchourron et al. 1985, Gardenfors 1988), we support two basic operations for updating an input specification: *revision* and *contraction*. The revision operation is used to incorporate new assertions or constraints into the input specification, admitting the possibility that the new input might contradict existing knowledge. The contraction operation is used to retract assertions or constraints (this is a non-trivial operation since there can be multiple possible ways of achieving the retraction, in general). The revision module thus performs update operations (either revisions or contractions) on the input specification to reflect changes in the environment, generating a consistent input specification as the outcome.

- **Scheduler Module:** The Scheduler module consists of three distinct routines that are invoked independently.
  - *Incremental Rescheduler:* This is invoked when a revised schedule becomes necessary to reflect relatively minor changes to the input specification. The intent is to exploit *dynamic constraint satisfaction techniques* to compute revised solutions which make maximal re-use of prior solutions (Mittal and Falkenheimer 1990, Hentenryck and Provost 1991, Dechter and Dechter 1994).
  - *Anytime Scheduler:* This is invoked in time-bounded applications and supports *anytime* scheduling algorithms. Anytime algorithms are a special class of real-time algorithms which satisfy the following three properties:
    * They are pre-emptively schedulable.
    * They may be interrupted at any time to obtain useful partial results.
    * They provide a formal guarantee that if they are interrupted at a later time point, the solution quality would be strictly better than that obtained at an earlier time point.

    There has been very little reported work on anytime constraint solving, and no work, to our knowledge, on anytime over-constrained solving (i.e., solving problems with both hard and soft constraints). The constraint-based representations obtained from the translation module are typically over-constrained specifications. Our approach therefore involves porting results from (Ghose and Goebel 1996) on anytime default reasoning to anytime over-constrained problem solving using connections between anytime reasoning and over-constrained problem solving identified in (Sattar, Ghose and Goebel, 1996).
  - *Full Scheduler:* The full-fledged scheduler is invoked in situations which are not stringently time-bounded, and where the deviation of the input specification from prior input specifications (if any) are significant enough to preclude any significant mileage being obtained from invoking the incremental rescheduler.

The scheduler module contains explicit mixed-initiative components. A simple case at point is the process through which the system discriminates between multiple competing schedules generated from an initially over-constrained specification in order to identify the preferred schedule (and hence the operative one). Even if the user were to specify, *a priori*, a prioritization of the input specification, this might still yield a set of preferred schedules (as opposed to a unique one). At this point the system would have to seek the user's input in order to identify which of the set of preferred schedules would be the operative one. The system provides further support to the process by presenting to the user *minimal experiments*, i.e., a minimal set of preferences sought from the user dynamically that would permit the identification of a unique preferred (or operative) schedule. If two schedules differ only in that one satisfies constraint A, but not constraint B, while the other satisfies B but not A, then a test to determine whether constraint A is preferred to constraint B would uniquely determine which of the two schedules is the preferred one.

4 The Specification Language

Any discussion of the RR-CONS architecture would be incomplete without at least a brief outline of the specification language, since the design of most of the significant components of the architecture have been driven by this choice of specification language. The RR-CONS specification language is a formal language which supports explicit notions of defeasible and non-defeasible (logical) assertions as well as hard and soft constraints. In the spirit of the integration between traditional logic programming and constraint-based reasoning achieved in the definition of the CLP Scheme (Jaffar and Lassez 1987), the RR-CONS specification language brings together default representations and constraint-based (especially over-constrained) specifications to define the expressively powerful notion of *constraint default theories* (Ghose 1997).

We shall provide below a brief formal outline of the specification language. It is important to note that we do not envisage scheduling systems based on the RR-CONS architecture insisting on the *formal* specification language being used at the level of the user interface. A pragmatic approach would be use a more user-friendly syntax, such as that of standard IF-THEN rule-based expert systems, and processing the inputs using the formal operations defined below. A *constraint default theory* is a pair (W, D) where W is a set of closed first-order formulae and D is a set of inference rules of the form:

$$\frac{:\beta(t_2)}{\gamma(t_3)}[]c_1(u_1), \ldots, c_n(u_n)$$

where u_i, t_i are vectors of terms in the domain DOM over which the constraints are defined, c_i are constraints defined in DOM and $\beta(t_2)$, $\gamma(t_3)$ are sentence schemas with β, γ being non- constraint predicate symbols. The constraint default rule above captures patterns of reasoning of the form: *if it consistent to assume* $\beta(t_2)$ *and each* $c_i(u_i)$ *together with the current set of accumulated constraints is solvable, then conclude* $\gamma(t_3)$. In a practical setting, this rule format could be used to specify a rule such as the following:

> *If it is consistent to assume that order O1 is ready to be shipped and that order O1 arrives at the loading dock at time t1 and that truck A may be loaded at time t2, then conclude that truck A must be loaded at time t2, provided the constraint t2=t1+5 may be satisfied.*

Notice that the applicability of this rule is contingent on several conditions: the order O1 being ready to ship (which may be violated by production delays), the order O1 arriving at the loading dock at time t1 (which may be violated by breakdowns in the materials handling system), the ability to load the truck at time t2 (which may be violated by a forklift that is out of order) and the constraint t2=t1+5 being satisfied (it may be violated if t1 takes on a value greater than t2). The formal machinery defined below provides a principled method of applying such rules while accounting for these complex conditions/assumptions. Given that each constraint default rule has the same fundamental interpretation as a default rule, there is clearly support for specifying defeasible assertions. In addition, the set of constraints associated with each constraint default rule permits the specification of soft constraints.

It is well-known (Reiter 1980) that default representations support multiple consistent, yet mutually exclusive views of the world (typically called *extensions*). Similarly, over-constrained specifications give rise to multiple, mutually exclusive, solutions. Similar considerations apply to constraint default logic theories, leading to the notion of constraint default extensions defined below.

Definition 1 *Let* (W, D) *be a constraint default theory. Let* E *and* J *be sets of formulae. Let* C *be a set of constraints.*

Define: $E_0 = W,\ J_0 = W,\ C_0 = \{\}$

and for $i \geq 0$*:*

$E_{i+1} = Cn(E_i) \cup \{\gamma(t'') \mid \frac{\alpha(t):\beta(t')}{\gamma(t'')}[]c_1(u_1), \ldots, c_i(u_i) \in D, J \cup \{\beta(t')\} \cup \{\gamma(t'')\} \not\vdash \perp, C_i \cup \{c_1(u_1), \ldots, c_i(u_i)\}$ *is solvable* $\}$

$J_{i+1} = Cn(J_i) \cup \{\beta(t') \wedge \gamma(t'') \mid \frac{\alpha(t):\beta(t')}{\gamma(t'')}[]c_1(u_1), \ldots, c_i(u_i) \in D, J \cup \{\beta(t')\} \cup \{\gamma(t'')\} \not\vdash \perp, C_i \cup \{c_1(u_1), \ldots, c_i(u_i)\}$ *is solvable* $\}$

$C_{i+1} = C_i \cup \{\{c_1(u_1, \ldots, c_i(u_i)\} \mid \frac{\alpha(t):\beta(t')}{\gamma(t'')}[]c_1(u_1), \ldots, c_i(u_i) \in D, J \cup \{\beta(t')\} \cup \{\gamma(t'')\} \not\vdash \perp, C_i \cup \{c_1(u_1), \ldots, c_i(u_i)\}$ *is solvable* $\}$

(E, J, C) *is a* constraint default extension *of* (W, D) *iff* $(E, J, C) = (\bigcup_{i=0}^{\infty} E_i, \bigcup_{i=0}^{\infty} J_i, \bigcup_{i=0}^{\infty} C_i)$.

In the definition above, the final extension that is computed consists of three data structures, which serve the following roles:

- E consists of a maximal set of conclusions that may be consistently held (these conclusions could be of the form *load truck A at time t2* etc.).
- J is a data structure used to ensure that the set of assumptions made in applying the rules that led to the conclusions contained in E are mutually consistent.
- C is a consistent set of constraints accumulated by applying the rules that led to the conclusions contained in E. Since procedures for testing the satisfiability of a set of constraints often generate a solution in the process, one may think of C as the solution to the classical constraint satisfaction problem consisting of these constraints.

One may think of the data structures E and C as representing the solution to a scheduling problem specified in this language, although one would actually use a translation to a different underlying constraint-based representation for such a solution.

5 Implementation/Future Work

An industry-scale implementation of a scheduling system for the house-frame manufacturing unit, based on the RR-CONS architecture, is currently underway. The system, called ISS (Intelligent Scheduling System), builds on the basic set of functionalities offered by the ILOG suite of constraint solvers. A preliminary prototype of the system currently exists and we expect to be able to report a more complete implementation in time for the conference. Much of our current work on ISS involves identifying and implementing heuristics for speeding up the process of generating satisfiable subsets of possibly over-constrained (and inconsistent) specifications. This is an important exercise, since the complexity of this process is the same as that for default inference, which is known to be NP-hard. Plans for future work include augmenting the system to include distributed scheduling capabilities, as well as advanced decision support functionalities such as what-if analysis.

6 References

Alchourron C., Gardenofrs P. and Makinson D., "On the logic of theory change: Partial meet functions for contraction and revision", *Journal of Symbolic Logic*, 50:510-530, 1985.

Davis G. and Fox M., "ODO: A constraint-based architecture for representing and reasoning about scheduling problems", *Proceedings of the Third Industrial Engineering Research Conference*, Atlanta, USA, 1994.

Dechter R. and Dechter A., "Structure driven algorithms for truth maintenance", *Artificial Intelligence*, 1994.

Fox M., *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*, Morgan Kaufman Publishers, Palo Alto, CA, US, 1987.

Gardenfors P., *Knowledge in Flux: Modelling the Dynamics of Epistemic States*, MIT Press, Cambridge, MA, USA, 1988.

Ghose A. K., Antoniou G., Sattar A., and Goebel R. G., "Connections between default reasoning and constraint satisfaction", *Information Science*, to appear, 1998.

Ghose A. K. and Goebel R. G., "Anytime default inference", *Proceedings of the 1996 Pacific Rim International Conference on Artificial Intelligence*, 1996.

Ghose A. K., "Non-monotonic constraint logic programming", unpublished manuscript, 1997.

Havens W. S., Ovans R., Sidebottom S., Sidebottom G. and Jones J., "Echidna: A constraint logic programming shell", *Proceedings of the Second Pacific Rim International Conference* *on Artificial Intelligence*, pages 165-171, 1992.

Hentenryck P. and Provost T., "Incremental search in constraint logic programming", *New Generation Computing*, 9:257-275, 1991.

Jaffar J. and Lassez J.-L., "Constraint logic programming", *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, Munich, Germany, 1987.

Jaffar J. and Maher M., "Constraint logic programming: A survey", *Journal of Logic Programming*, 19-20, 1994.

Jaffar J., Michaylov S., Stuckey P. and Yap R., "The clp(r) language and system", *ACM Transactions on Programming Languages and Systems*, 14(3), 1992.

Keng N. and Yun D., "A planning/scheduling methodology for the constrained resource problem", *Proceedings of the 1989 International Joint Conference on Artificial Intelligence*, 1989.

Minton S., Johnston M. D., Philips A. B., Laird P., "Minimizing conflicts: A heuristic repair method for constraint satisfactio and scheduling problems", *Artificial Intelligence*, 58(1-3):161-205, 1992.

Mittal S. and Falkenheimer B., "Dynamic constraint satisfaction problems", *Proceedings of the Eighth National Conference on Aritificial Intelligence*, pages 25-32, 1990.

Poole D., Goebel R. G. and Aleliunas R., "Theorist: A logical reasoning system for defaults and diagnosis", *The Knowledge Frontier: Essays in the Representation of Knowledge* (Cercone N. J. and McCalla G., eds.), pages 331-352, Springer Verlag, 1987.

Reiter R., "A logic for default reasoning", *Artificial Intelligence*, 13:81-132, 1980.

Sadeh N., *Lookahead Techniques for Micro-Opportunistic Job Shop Scheduling.* PhD thesis, Carnegie Mellon University, USA, 1991.

Sattar A., Ghose A. K. and Goebel R. G., "Specifying over-constrained problems in default logic", *Over-constrained systems* (Jampel M., Freuder E. and Maher M., editors), Springer Verlag, 1996.

Sidebottom G. and Havens W. S., "Nicolog: A simple yet powerful cc(fd) language", *Journal of Automated Reasoning*, to appear.

Zweben M., Davis E., Daun B. and Deale M., "Iterative repair for scheduling and rescheduling", *IEEE Transactions on Systems, Man and Cybernetics.* 1993.

Zweben M., Davis E., Daun B., Drascher E., Deale M. and Eskey M., "Learning to improve constraint-based scheduling", *Artificial Intelligence*, 58:271-296, 1992.

Zweben M., Deale M. and Garan R., "Anytime rescheduling", *Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning*, 1990.

Zweben M. and Fox M., *Intelliegnt Scheduling*, Morgan Kaufman, Palo Alto, CA, USA, 1994.