Appeared In: Proceedings of IJCAI '93 Workshop on Production Scheduling

ODO: A Constraint-based Architecture for Representing and Reasoning About Scheduling Problems

Eugene Davis*
Department of Computer Science
University of Toronto
Toronto, Ontario, M5S 1A4, Canada
gdavis@cs.utoronto.ca

Mark Fox
Department of Industrial Engineering
University of Toronto
Toronto, Ontario M5S 1A4, Canada
msf@ie.utoronto.ca

Abstract

We present work-in-progress on ODO, a constraint-based scheduling architecture. ODO employs both constructive and iterative search methods, and bases heuristic decisions on problem property measures (*textures*). With the architecture's command language a user specifies a problem to be solved and the search parameters used in solving the problem. We plan to use ODO to study relationship between problem textures and efficient search heuristics for both generative and iterative scheduling methods.

Introduction

In the past few years, research in knowledge-based approaches to scheduling has focused on graph-based constraint satisfaction and optimization techniques [Sadeh, 1991; Zweben *et al.*, 1992; Keng and Yun, 1989]. In this approach, a problem is represented by a constraint graph, where the nodes are the variables of tasks and resources, and the arcs are constraints

*Supported by a University of Toronto Open Fellowship and a Research Assistanceship from the Department of Computer Science

among the variables. Solving a problem amounts to assigning values to all variables such that all constraints are satisfied.

There are two common methods for solving scheduling problems represented in this way. The *constructive* method [Fox, 1987; Sadeh, 1991; Keng and Yun, 1989] starts with an empty schedule and assigns a value to a variable only if it is consistent with all previous assignments; if the current set of assignments cannot lead to a feasible solution, then the method backtracks and tries again. The *iterative* method [Zweben *et al.*, 1993b; Minton *et al.*, 1992] starts with values assigned to all variables and repeatedly modifies those values until all constraints are satisfied.

Despite their differences, the constructive and iterative problem solvers have at least one similarity: both methods continually modify the current schedule in a *search* to find a solution as quickly as possible. Given that search could in the worst case take exponential time, it becomes important to select appropriate modifications in an efficient manner. The term *heuristic* is used to describe a modification scheme that on average performs well.

The successful search heuristic usually exploits structural properties of the given problem. We use the term *textures* [Fox *et al.*, 1989] to describe these properties when the problem is represented in a constraint model. To date, little work has been done to explore the relationship between problem textures and the efficiency of problem solving methods.

We are interested in questions related to textures and efficient search for scheduling problems: What are the textures of this domain? How might textures be combined? Can we correlate textures with good heuristics? What is the relationship between constructive and iterative search? How does problem reformulation (abstraction, aggregation) change the way we solve the problem?

As a platform for exploring these issues, we are building a generic scheduling architecture, ODO, which combines constructive and iterative scheduling approaches, and employs a texture library with which search heuristics will base their decisions. The architecture includes a command language for declaring problem instances, performing texture measures, and controlling search parameters. In the paper we further discuss the notions of constraint representation, textures, and search, and describe our design of the scheduling architecture.

Constraint Representation

We see several reasons for the success of constraint-based representation in the scheduling domain. First, a constraint model is a natural representation: scheduling is a decision problem that can be described with a finite number of variables, each with a finite domain. Second, since the scheduling problem is dynamic in nature, the addition and deletion of activities, machines, deadlines, etc., can be easily realized by adding and deleting appropriate variables and constraints. Third, a number of tools exists to manipulate constraint graphs [Mackworth, 1977; Dechter and Meiri, 1989] that apply particularly well to scheduling problems [Smith, 1983; Le Pape and Smith, 1987]. Also, as mentioned before, two powerful search methods (constructive and iterative) can be efficiently performed in this framework.

In ODO's constraint model, problems are represented by a collection of objects, variables,

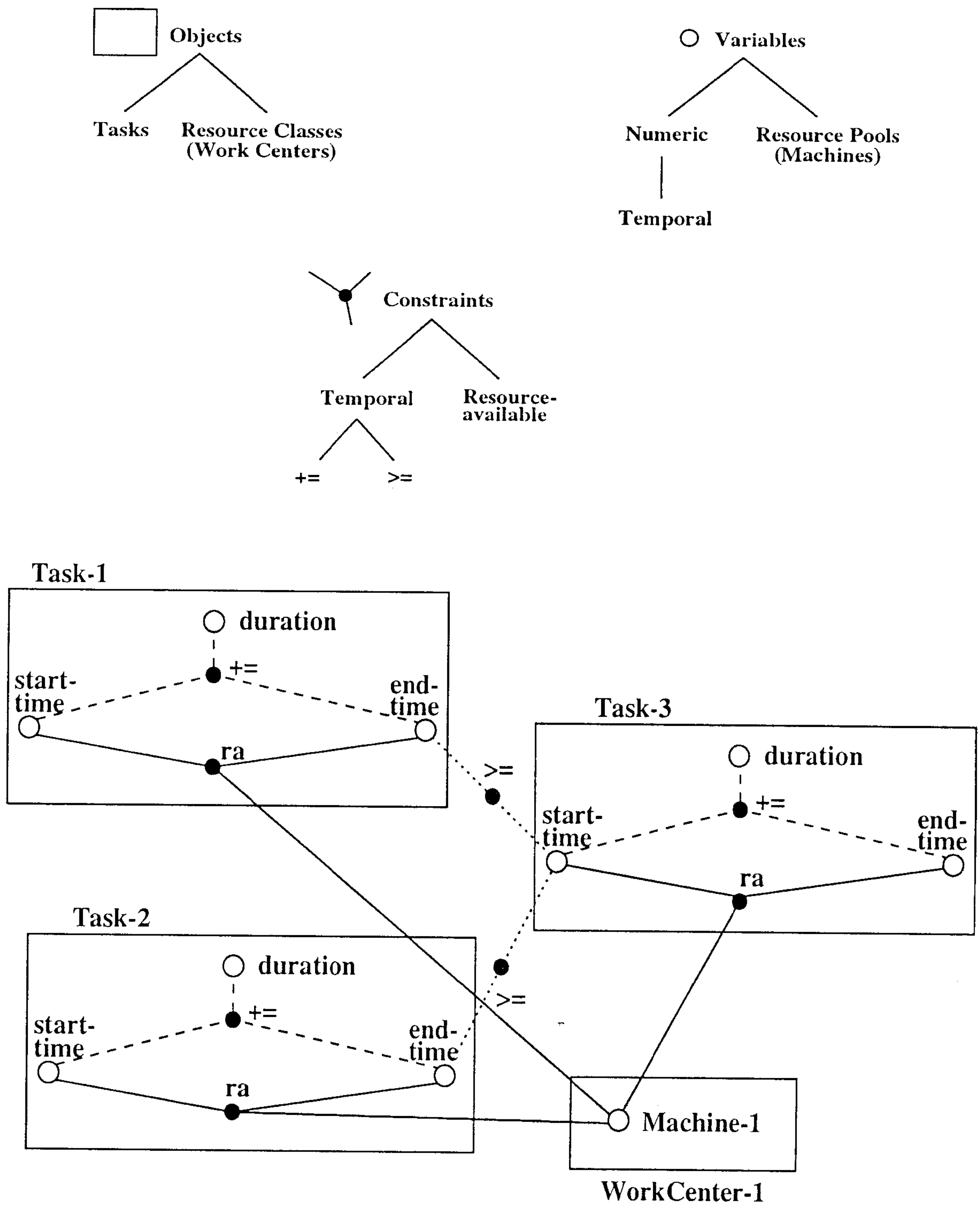

Figure 1: An ODO problem hierarchy and constraint network

and constraints. The objects serve as placeholders for variables and constraints, and may be used to store measured texture information. Figure 1 presents a hierarchy adequate to represent simple job-shop scheduling problems, along with a contention graph for a simple problem instance. As noted in the hierarchy, objects are represented as boxes, variables as hollow circles, and constraints as filled circles with lines drawn to the relevant constrained variables.

The problem we initially address is job-shop scheduling with due dates. Though there exist constraint representations for more complex constraints (see for example [Zweben *et al.*, 1993a]), we will initially restrict our attention to precedence and resource constraints.

Textures

A texture is a property of a constraint graph. Because some textures require exponential computation to compute, we usually estimate their actual value, hoping that the estimate is close. These texture estimates are what the heuristics are a function of. The following are examples:

- In backtracking search, the next schedule modification is chosen that will least likely cause backtracking to occur. In addition, modifications are ordered such that if backtracking will occur, it will do so as soon as possible in order to minimize thrashing. In the constraint model, this modification principle can be summarized as follows: find the most constrained variable, and assign it a value that least constrains all later assignments. In MICROBOSS [Sadeh, 1991], for example, the most constrained variable is the activity that relies upon the most contended resource/time reservation, and the least constraining value is that reservation estimated as having the highest probability not to conflict with later activity-resource/time assignments.

- In iterative search, the next modification is chosen that will hopefully reduce the number of violated constraints by the greatest amount. One approach, a variant of MIN-CONFLICTS [Minton *et al.*, 1992], selects the activity participating in the most number of violations and moves it to the time that would result in a schedule with the fewest number of overall violations.

The success of these heuristics in their respective domains suggests that the heuristics should perform equally well on problems with similar textures. In Fox and Sadeh's initial paper on textures [Fox *et al.*, 1989], the authors define *variable/value goodness* and *variable tightness* textures and show how they relate to the least-constraining value and most-constrained variable concepts, respectively. *Variable/value goodness* is defined as "the probability that the assignment of a particular variable to a particular value will lead to an overall solution." *Variable tightness* is defined as "the probability that an assignment consistent with all the problem constraints that do not involve that variable does not result in a solution."

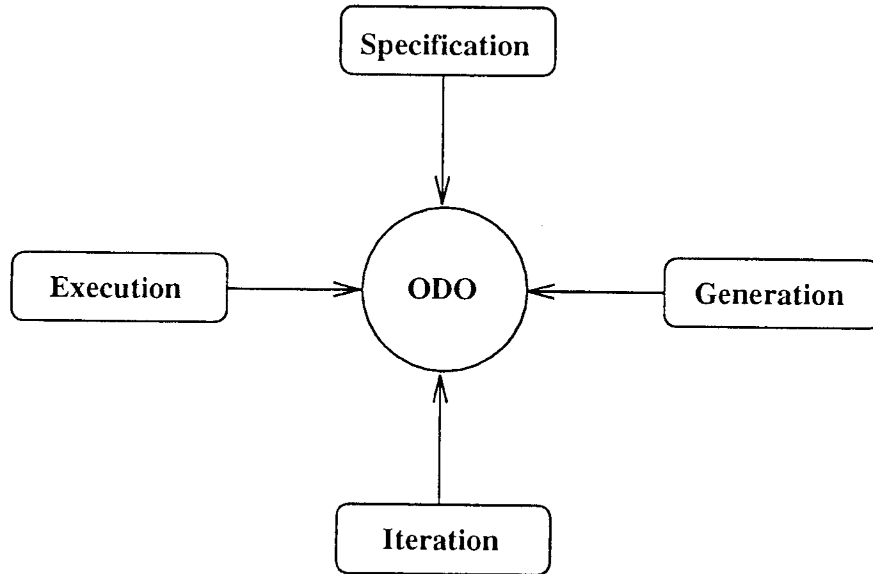

Figure 2: ODO architecture

Our intent is to characterize a larger set of textures than those identified in [Fox *et al.*, 1989], identify close and efficient texture estimators, and show how scheduling heuristics are functions of these textures for both generative and iterative cases.

Search Techniques

Within the constraint-based framework, we assert that the operations performed by a scheduler can be classified into one of the following functional *phases*:

- Specification - declaring the problem's variables and constraints
- Generation - assigning values to variables
- Iteration - changing values on variables
- Execution - assigning the actual values to variables

Typically these phases are encountered in a cycle. First a problem is specified, then a solution is generated and improved upon, and this solution is executed. We generalized upon the notion of looping through the functional phases by making ODO capable of performing any phase at any time (see Figure 2). This makes it straightforward, for example, to *incrementally* specify and solve small parts of a large problem.

The phase decomposition captures the functionality of both constructive and iterative search paradigms. Even though it is quite conceivable to interleave generation and iteration steps, most well-known systems focus search in one phase until a solution is reached.

The search performed may be systematic or nonsystematic. If systematic, all variable assignment possibilities are eventually considered: if the search terminates with no solution found, it is known that no solution exists. Nonsystematic approaches do not eventually exhaust the search space (or at least are not aware of that fact if they do). Since they do not maintain information to perform a methodical search, they can more easily move about the search space. However, they may also visit the same search state many times, and hence cycle. Constructive approaches are typically systematic[1], and iterative approaches are nonsystematic.

In the systematic constructive approach, a backtracking algorithm [Golomb and Baumert, 1965] is typically employed. Researchers have identified many enhancements to the basic backtrack algorithm in one of several algorithm components [Haralick and Elliott, 1980; Dechter and Pearl, 1988; Ginsberg *et al.*, 1990; Bitner and Reingold, 1975; Gaschnig, 1977; Sadeh, 1991]:

- Preprocessing - (e.g. removing symmetrical states from consideration)
- Variable Selection - (e.g. most constrained, most constraining)
- Value Selection - (e.g. least constraining)
- Constraint Propagation - (e.g. forward-checking, arc-consistency)
- Backtracking Mechanisms - (e.g. dependency-directed, backmarking, backjumping)

The iterative approach also has many options [Minton *et al.*, 1992; Zweben *et al.*, 1993b]:

- Initial Solution Generation - (e.g. CPM)
- Variable Selection - (e.g. most violated variable)
- Value Selection - (e.g. value resulting in the least number of violations)
- Constraint Propagation - (e.g. forward-checking, arc-consistency)
- Intermediate Solution Acceptance Criteria - (e.g. strict hill climbing, simulated annealing)

Whether constructive or iterative, any assignment could be thought of in terms of making a modification to the existing scheduling state. These modifications continue in a cycle (perhaps with the occasional backtrack or rejection of a solution), until some termination condition is satisfied. In ODO we plan to model all search this way. Figure 3 captures our perception of this loop.

[1] We note that the systematic approach may not always be best. In [Langley, 1992] the author shows that a nonsystematic constructive search can outperform a systematic one.

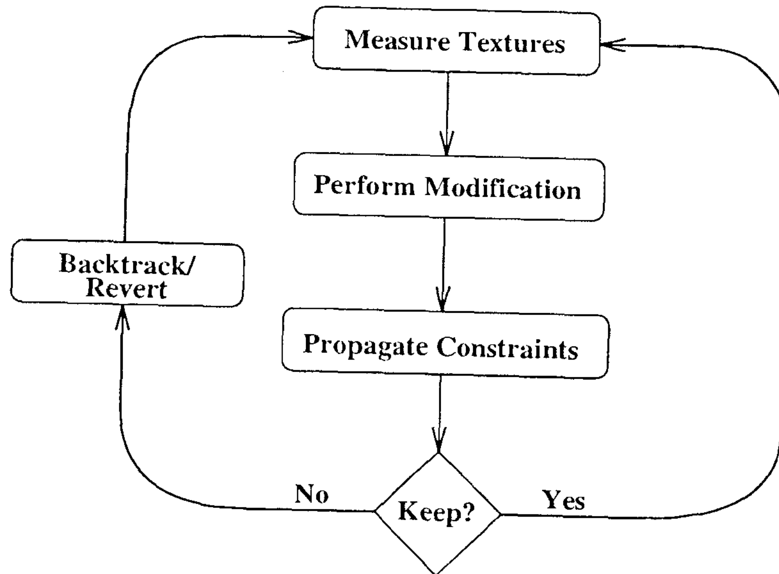

Figure 3: Flow chart of the problem solving component

Problem Specification and Execution

During the specification phase of ODO, the problem is described. Through the built-in language (see below) the user defines tasks, resources, and temporal constraints. In addition, the user can add or remove variables and constraints, and preassign values to variables.

In the execution phase, the scheduler accepts as input an executable schedule, and uses that to dispatch activities. It responds to real-world events; however, its flexibility is restricted to the *slack* provided in the schedule. If the execution phase cannot use the schedule as provided, then the schedule should be sent back to the iteration phase for repair.

Built-in Language

We have devised a simple command language as a user interface to ODO. As commands are parsed from the input, the appropriate actions are called. When used this way ODO could be thought of as an interpreter. With this language the user can completely describe the problem instance. In addition, the user controls which textures to measure and which search parameters to use.

The language interface is a module; as such it could be replaced with a graphical interface without changing the architecture's functionality. In addition, the language module could be connected to a new module designed to generate compilable source code directly from the parsed commands.

The following is an example of how the language might be used in a simple problem that encounters the specification, generation, and iteration phases. The problem specified

is compatible with that found in Figure 1, and the problem solving mechanism emulates a version of MIN-CONFLICTS.

```
resource-class WorkCenter-1
resource Machine-1 WorkCenter-1
task Task-1 100
task Task-2 100
task Task-3 100
constraint temporal-after Task-1 Task-3
constraint temporal-after Task-2 Task-3
constraint resource-dedicated Task-1 WorkCenter-1
constraint resource-dedicated Task-2 WorkCenter-1
constraint resource-dedicated Task-3 WorkCenter-1
set ALL-TASKS Task-1 Task-2 Task-3
assign ALL-TASKS earliest-time
var-heuristic MAX-C most-violated-var random
val-heuristic MIN-C least-violated-val random
arc-consistency-temporal ALL-TASKS
while cost >= 0 do
    iterate ALL-TASKS MAX-C MIN-C lookahead-1 hill-climbing
```

The above statements define the following: three tasks and one resource are declared, the tasks are given duration 100, and the domains of all of the tasks are pruned by an arc-consistency algorithm [Mackworth, 1977] on the temporal constraints. All task variables are assigned values that are earliest in their domains. Finally, iterative-repair is called until no constraints remain violated, selecting variables by the maximum number of conflicts (breaking ties randomly) and values by the locations resulting in the least number of violations (breaking ties randomly).

Summary, Status, and Future Work

Figure 4 gives an overall view of ODO's constraint model within its several phases. As a problem is specified, it is represented as a constraint graph. In the generation and iteration phases, a search is performed (each square in the search tree represents the constraint graph at that point in the search), and textures are measured (and cached on the constraint graph) as desired. If ODO finds a solution, it is passed to the execution phase, which will respond to real-time events within the solution's slack parameters.

Our first goals are to create a library of texture measures and heuristics, and to explore the tradeoff between generative and iterative scheduling. ODO will be used to model scheduling activity for the TOVE (Toronto Virtual Enterprise) project [Fox, 1992], which aims to model dynamic commercial enterprises in a software environment.

One concern we have is to make the utility of ODO as insensitive as possible to implementation issues (such as whether or not to represent a constraint network in a matrix or

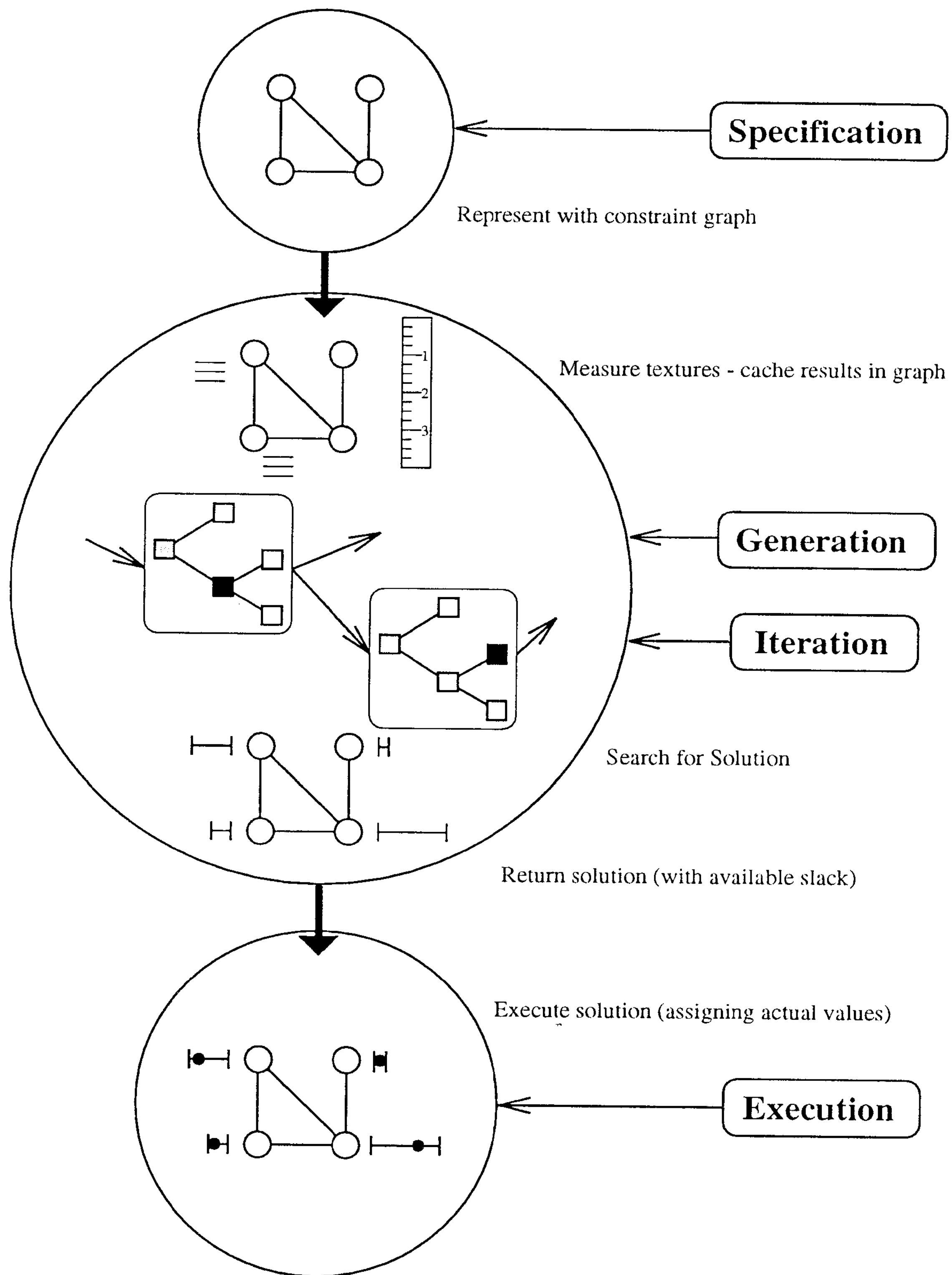

Figure 4: Versions of the constraint model within ODO

as an array of linked lists). If this cannot be avoided, we may incorporate necessary details into the architecture's command language.

In the future we plan to enhance ODO so that it can also represent schedule abstractions. Eventually we hope to modify ODO to become testbed for machine-learning techniques in heuristic selection and automated schedule abstraction.

Related Work

The principles of providing the user with a declarative programming language for use within a constraint-based problem solver can be found Van Hentenryck's CHIP system and Minton's MULTI-TAC [Minton, 1993]. CHIP extends logic programming to reason more explicitly about constraints and to give the programmer more control over the type of backtrack search the problem solver should perform. MULTI-TAC takes as input a description of a combinatorial problem and generates an appropriate problem-solver. Both of these currently only attempt to solve problems with constructive approaches, although the designers of MULTI-TAC plan to add an iterative component in the near future.

In [Le Pape, 1991], the author summarizes research in the utility of various constraint propagation and backtracking techniques in the domain of job-shop scheduling, and presents an architecture for the interaction of a predictive scheduler (our "generator") and a reactive dispatcher (our "executor").

Finally, the constraint-based model used in ODO is based upon that found in GERRY[Zweben *et al.*, 1993b] and MICROBOSS[Sadeh, 1991].

References

[Bitner and Reingold, 1975] J. Bitner and E. Reingold. Backtrack programming techniques. *Communications of the ACM*, 18(11):651–656, November 1975.

[Dechter and Meiri, 1989] R. Dechter and I. Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *Proceedings of IJCAI-89*, 1989.

[Dechter and Pearl, 1988] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.

[Fox *et al.*, 1989] M. Fox, N. Sadeh, and C. Baykan. Constrained heuristic search. In *Proceedings of IJCAI-89*, 1989.

[Fox, 1987] M. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* Morgan Kaufman Publishers, Inc., 1987.

[Fox, 1992] M. Fox. The TOVE Project: Towards a Common Sense Model of the Enterprise. Technical report, Department of Industrial Engineering, University of Toronto, April 1992.

[Gaschnig, 1977] J. Gaschnig. A general backtrack algorithm that eleminates most redundant tests. In *Proceedings of IJCAI-77*, 1977.

[Ginsberg *et al.*, 1990] M. Ginsberg, M. Frank, M. Halpin, and M. Torrance. Search lessons learned from crossword puzzles. In *Proceedings of AAAI-90*, 1990.

[Golomb and Baumert, 1965] S. Golomb and L. Baumert. Backtrack programming. *Journal of the ACM*, 12(4):516–524, 1965.

[Haralick and Elliott, 1980] R. Haralick and G. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[Keng and Yun, 1989] N. Keng and D. Yun. A planning/scheduling methodology for the constrained resource problem. In *Proceedings of IJCAI-89*, 1989.

[Langley, 1992] P. Langley. Systematic and nonsystematic search strategies. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, 1992.

[Le Pape and Smith, 1987] C. Le Pape and S. Smith. Management of temporal constraints for factory scheduling. Technical Report CMU-RI-TR-87-13, Robotics Laboratory, Carnegie Mellon University, June 1987.

[Le Pape, 1991] C. Le Pape. Constraint propagation in planning and scheduling. Technical report, Robotics Laboratory, Stanford Universtity, January 1991.

[Mackworth, 1977] A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Minton *et al.*, 1992] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[Minton, 1993] S. Minton. Integrating heuristics for constraint satisfaction problems: A case study. In *Proceedings of AAAI-93*, 1993.

[Sadeh, 1991] N. Sadeh. *Lookahead Techniques for Micro-Opportunistic Job Shop Scheduling.* PhD thesis, Carnegie Mellon University, 1991. CMU-CS-91-102.

[Smith, 1983] S. Smith. Exploiting temporal knowledge to organize constraints. Technical report, The Robotics Institute, Carnegie Mellon University, 1983.

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

[Zweben *et al.*, 1993a] M. Zweben, E. Davis, B. Daun, and M. Deale. Informedness vs. computational cost of heuristics in iterative repair scheduling. In *Proceedings of IJCAI-93*, 1993.

[Zweben *et al.*, 1993b] M. Zweben, E. Davis, B. Daun, and M. Deale. Iterative repair for scheduling and rescheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, To Appear, 1993.