ODO: A CONSTRAINT-BASED ARCHITECTURE FOR REPRESENTING AND REASONING ABOUT SCHEDULING PROBLEMS

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

Mark S. Fox
Department of Industrial Engineering
University of Toronto
Toronto, Ontario, CANADA M5S 1A4
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 the 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 [18] [22] [11]. 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 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 [18] [4] [11] 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 [20] [17] 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 th on average performs well.

The successful search heuristic usually exploits structural properties of the given problem. We use the term *textures* [to describe these properties when the problem is represente in a constraint model. To date, little work has been done t 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 texture of this domain? How might textures be combined? Can w correlate textures with good heuristics? What is the relatio ship between constructive and iterative search? How does problem reformulation (abstraction, aggregation) change th way we solve the problem?

As a platform for exploring these issues, we are building a generic scheduling architecture, ODO, which combines cor structive and iterative scheduling approaches, and employs 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 [15] [2] that apply particularly well to scheduling

INSTITUTE OF INDUSTRIAL ENGINEERS
3rd Industrial Engineering Research Conference Proceedings

problems [19] [14]. 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, 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.

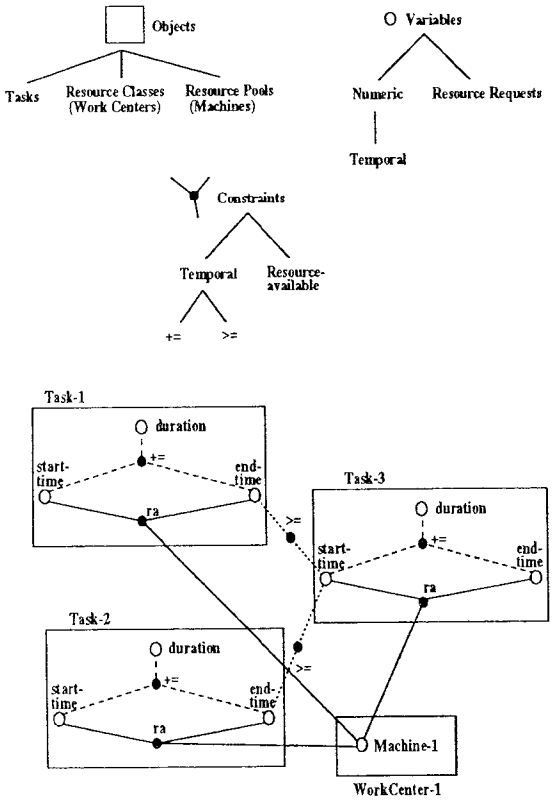

Figure 1. Problem constraint hierarchy and sample constraint network.

The problem we initially address is job-shop scheduling with due dates. Though there exist constraint representations for more complex constraints (see for example [22]), we will initially restrict our attention to precedence and resource constraints.

TEXTURES

A texture is a property of a constraint graph. Because some texture measures require exponential computation, 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 [18], 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 [17], 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 [6] 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."

Our intent is to characterize a larger set of textures than those identified in [6], 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 variable

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.

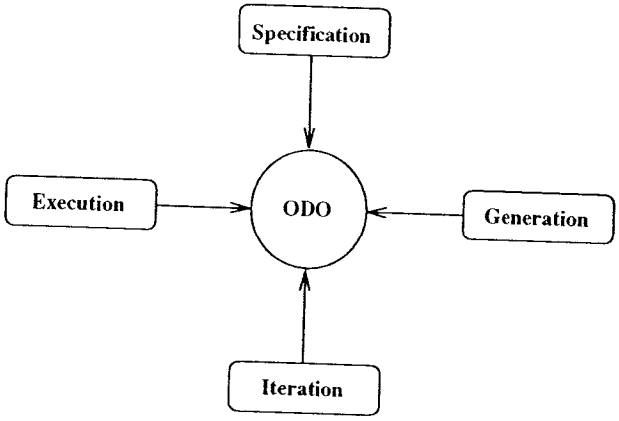

Figure 2. ODO Architecture.

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.

1. We note that the systematic approach may not always be best. In [12] the author shows that a nonsystematic constructive search can outperform a systematic one.

In the systematic constructive approach, a backtracking algorithm [9] is typically employed. Researchers have id tified many enhancements to the basic backtrack algorith in one of several algorithm components [3] [10] [8] [1] [[18]:

- Preprocessing - (e.g. removing symmetrical states fro consideration)
- Variable Selection - (e.g. most constrained, most constraining)
- Value Selection - (e.g. least constraining)
- Constraint Propagation - (e.g. forward-checking, arc-c sistency)
- Backtracking Mechanisms - (e.g. chronological, backmarking, backjumping)

The iterative approach also has many options [17] [20]:

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

The specific impact of choosing one particular search optio over another is not well known, and is a focus of our research. Our first step is to provide a structure to the searc process from which the seach options can be utilized. Whether constructive or iterative, any assignment can be viewed 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 solu tion), until some termination condition is satisfied. In ODO we plan to model all search this way. Figure 3 captures our perception of this loop

Constraint-based problem solving can be viewed as perform ing search in the manner described above; the details of the search process are explicitly represented by the parameters to the search process: how to perform variable and value selection, how much constraint proagation, etc. ODO will structure search in this way; thus we will be able to associate heuristic performance with the actual problem properties and problem solving parameters from within our model.

PROBLEM SPECIFICATION AND EXECUTION

The problem is declared in the specification phase of ODO. 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.

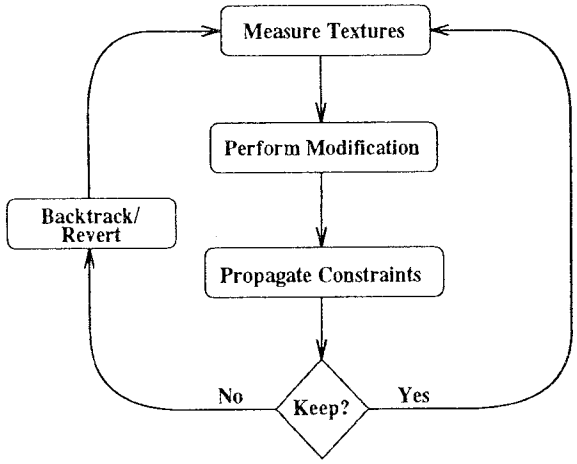

Figure 3. Flow chart of the problem solving component.

ODO 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;
before Task-1 Task-3;
before Task-2 Task-3
resource_request Task-1
 WorkCenter-1 1;
resource_request Task-2
 WorkCenter-2 1;
resource_request Task-3
 WorkCenter-3 1;
enforce arc_consistency temporal;
assign_all task_times random;
assign_all task_resource_pools
 arbitrary;
var_selection violated random;
val_selection all;
evaluation_method cost_lookahead_1;
evaluation_criteria min_value random;
accept_criteria always;
while ((cost > 0) && (search_time
 < 150)) do repair;
```

SUMMARY, STATUS, AND FUTURE WORK

In this paper we have presented our model for constraint-based scheduling. This model is based upon our analysis of the common components of constraint-based problem solvers. We have reviewed some of these components and have described how we anticipate integrating them into a generic scheduling system, ODO. As a problem is specified, it is represented as a constraint graph. In the generation and iteration phases, 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 parametersVersions of the Constraint Model in ODO

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 [5], 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 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 [16]. 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 [13], 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 [20] and MicroBOSS [18].

REFERENCES

[1] Bitner, J. and Reingold, E. Backtrack Programming Techniques. *Communications of the ACM*. 18(11):651-656, November, 1975.

[2] Dechter, R. and Meiri, I. Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems. *Proceedings of IJCAI-89*. 1989.

[3] Dechter, R. and Pearl, J. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*. 341-38, 1988.

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

[5] Fox, M. *The TOVE Project: Towards a Common Sense Model of the Enterprise*. Technical Report, .

[6] Fox, M. and Sadeh, N. and Baykan, C. Constrained Heuristic Search. *Proceedings of IJCAI-89*. 1989.

[7] Gaschnig, J. A General Backtrack Algorithm that Eleminates Most Redundant Tests. *Proceedings of IJCAI-77*. 1977.

[8] Ginsberg, M. and Frank, M. and Halpin, M. and Torrance, M. Search Lessons Learned from Crossword Puzzles. *Proceedings of AAAI-90*. 1990.

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

[10] Haralick, R. and Elliott, G. Increasing Tree Search ciency for Constraint Satisfaction Problems. *Artificial In ligence*. 14 263-313, 1980.

[11] Keng, N. and Yun, D. A Planning/Scheduling Met ology for the Constrained Resource Problem. *Proceedin IJCAI-89*. 1989.

[12] Langley, P. Systematic and Nonsystematic Search Strategies. *Proceedings of the First International Confere on Artificial Intelligence Planning Systems*. 1992.

[13] Le Pape, C. *Constraint Propagation in Planning an Scheduling*. Technical Report, Robotics Laboratory, Stanf Universtity, January, 1991.

[14] Le Pape, C. and Smith, S. *Management of Tempora Constraints for Factory Scheduling*. Technical Report CM RI-TR-87-13, Robotics Laboratory, Carnegie Mellon Uni versity, June, 1987.

[15] Mackworth, A. Consistency in Networks of Relatio *Artificial Intelligence*. 89-118, 1977.

[16] Minton, S. Integrating Heuristics for Constraint Sati faction Problems: A Case Study. *Proceedings of AAAI-93* 1993.

[17] Minton, S. and Johnston, M. and Philips, A. and Lai P. Minimizing conflicts: a heuristic repair method for con straint satisfaction and scheduling problems. *Artificial Int ligence*. 58 161-205, 1992.

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

[19] Smith, S. *Exploiting Temporal Knowledge to Organ Constraints*. Technical Report, The Robotics Institute, Ca egie Mellon University, 1983.

[20] Zweben, M. and Davis, E. and Daun, B. and Deale, Iterative Repair for Scheduling and Rescheduling. *IEEE Transactions on Systems, Man, and Cybernetics*. Decemb 1993.

[21] Zweben, M. and Davis, E. and Daun, B. and Drasch E. and Deale, M. and Eskey, M. Learning to improve constraint-based scheduling. *Artificial Intelligence*. 58 271-29 1992.

[22] Zweben, M. and Davis, E. and Daun, B. and Deale, Informedness vs. Computational Cost of Heuristics in Iter tive Repair Scheduling. *Proceedings of IJCAI-93*. 1993.

BIBLIOGRAPHICAL SKETCH

Eugene Davis is a Ph.D. student in Computer Science at the Universtiy of Toronto. Before returning to school he helped design, develop, and support a constraint-based scheduling system used at NASA in the preparation of Space Shuttle orbiters for launch. He has published papers in *Artificial Intelligence* and *IEEE Transactions on Systems, Man, and Cybernetics* journals.

Dr. Mark Fox is Professor of Industrial Engineering, Computer Science and Management Science at the University of Toronto. He holds positions as NSERC Industrial Research Chairholder in Enterprise Integration, and Director of the Collaborative Program in Integrated Manufacturing. His work in constraint-directed scheduling has led to the creation of several commercially successful scheduling systems and the initiation of the field of Knowledge-Based Scheduling. His current research interests include enterprise integration, constraint directed reasoning, a unified theory of constraint-directed scheduling and their application to engineering and manufacturing problems such as Concurrent Engineering, Supply Chain Management and Enterprise Design. Dr. Fox was elected a Fellow of American Assoociation for Artificial Intelligence in 1991, and a Joint Fellow of the Canadian Institute for Advanced Research and PRECARN in 1992. He is a past AAAI councilor, and a member of ACM, IEEE, SME, CSCSI, IIE and TIMS. Dr. Fox has published over 60 papers.