CHS-Soar: Introducing Constrained Heuristic Search to the Soar Cognitive Architecture

Sean A. Bittle[1], Mark S. Fox[2]

Department of Mechanical and Industrial Engineering, University of Toronto,
King's College Road, Toronto, Ontario, Canada, M5S 3G9
sean.bittle@utoronto.ca[1], msf@mie.utoronto.ca[2]

Abstract

Cognitive architectures aspire for generality both in terms of problem solving and learning across a range of problems, yet to date few examples of domain independent learning has been demonstrated. In contrast, constraint programming often utilizes the same domain independent heuristics to find efficient solutions across a broad range of problems types. This paper provides a progress report on how a specific form of constraint-based reasoning, namely Constrained Heuristic Search (CHS) can be effectively introduced into an integrated symbolic cognitive architecture (Soar) to achieve domain independent learning. The integration of CHS into Soar retains the underlying problem-solving generality of Soar, yet exploits the generalized problem representation and solving techniques associated with constraint programming. Preliminary experiments are conducted on two problems types: Map Colouring and Job Shop Scheduling, both of which are used to demonstrate a domain independent learning using texture based measures.

Introduction

Cognitive architectures specify the underlying infrastructure of tightly coupled mechanisms that support the acquisition and use of knowledge. They aspire to demonstrate problem-solving capabilities ranging from solving highly routine to more difficult problems using large bodies of diverse knowledge [Laird, 2008]. Research on cognitive architectures is important because it supports a central goal of artificial intelligence - namely the creation and understanding of artificial agents that demonstrate similar problem-solving capabilities to humans [Langley et. al., 2006]. While AI research over the past two decades has successfully pursued specialized algorithms for specific problems, cognitive architectures aim for breadth of coverage across a diverse set of tasks and domains [Laird, 2008].

However, to date there appears to be few examples of effective problem-solving or domain independent learning on realistic problems by symbolic cognitive architectures [Diaper, et. al, 2007]. In contrast, Constraint Programming (CP) has for years established its ability to find efficient solutions to a broad domain of specific real-world problems using domain independent heuristics [Wallace, 1996]. CP, however, has not generally promoted itself as the central problem-solving approach or learning framework for cognitive architectures.

Symbolic cognitive architectures often employ rule-based deductive reasoning to encode knowledge and guide problem solving and learning. Often the division between problem representation and problem solving is often co-mingled resulting in agents that can only solve a specific problem type and often preclude any form of domain independent learning. In contrast, constraint programming employs a different, yet complementary form of deductive reasoning based on a generalized problem representation which allows it to utilize generic problem solving techniques such as constraint propagation.

Our work seeks to integrate these two different problem-solving paradigms[1] (constraint and rule-based reasoning) into a unified, declarative architecture; namely to introduce Constrained Heuristic Search (CHS) [Fox et. al, 1989] to the Soar cognitive architecture [Laird et. al, 1987]. This paper reports progress to date demonstrating a specific form of domain independent learning. Our broader research effort posits that CHS can provide the requisite framework for unifying the features of constraint programming and symbolic cognitive architectures in order to demonstrate practical problem solving performance and domain independent learning with cognitive architectures.

Soar Cognitive Architecture

One of the earliest symbolic cognitive architectures developed was Soar, created by John Laird, Allen Newell, and Paul Rosenbloom at Carnegie Mellon University [Laird et. al. 1987]. Soar is a symbolic architecture for general intelligence that integrates basic mechanisms for

[1] In unpublished work by Shivers, the idea of introducing a form of constraint propagation in Soar was suggested [Shivers, 1986].

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

problem solving, use of knowledge, learning, and perceptual-motor behaviour [Laird et al., 1987].

Soar consists of a single long-term memory encoded as production rules and a single transient declarative working memory. All long-term knowledge in Soar is represented as if-then rules, called productions or production rules. Soar's symbolic working memory holds the agent's assessment of the current situation derived from perception and retrieval of knowledge from its long-term memory [Laird, et. al. 1987]. Working and long-term memories are linked through a decision cycle which selects operators to change states and detects impasses. Problem solving is driven by the act of selecting problem spaces, states, and operators in which each selection is accomplished through a three-phase decision cycle. The phases are repeated until the goal of the current task has been achieved. Chunking is Soar's learning mechanism that converts the results of problem solving in subgoals into rules.

Constraint Programming

Independently, yet in close chronological parallel to the problem-solving work of Newell and Simon, an alternative problem-solving approach evolved called Constraint Programming (CP). Constraint satisfaction is a sub-domain of constraint programming dealing with problems defined over a finite domain and involve no objective function. More formally, a Constraint Satisfaction Problem (CSP) is defined as a set of variables; for each variable, a finite set of possible values (its domain), and a set of constraints restricting the values that the variables can simultaneously assume [Kumar, 1992]. Variables are linked by constraints that can be either logical (i.e. $X \leq Y$) or symbolic (i.e. *X Loves Y).*

The principal techniques utilized to solve a CSP include a combination of constraint propagation, variable and value ordering and search. Constraint propagation is the term for propagating the implications of a constraint on one variable onto other variables [Kumar, 1992]. The objective of propagation is to reduce the size of the variable domains based on the constraints imposed on them. Variable and value ordering heuristics can be utilized to dramatically improve the efficiency of search and suggest which variable should be assigned next and in what order the values should be tried [Kumar, 1992]. The role of constraints in problem solving has been reviewed by Fox, where he notes constraints can provide structure to search thereby reducing the size of the search space [Fox, 1986].

Constrained Heuristic Search The idea of augmenting the definition of the problem space through the introduction of a constraint graph representation is encompassed in the Constrained Heuristic Search (CHS) problem-solving model developed by Fox [Fox et. al., 1989]. CHS is a combination of constraint satisfaction and heuristic search. In this model, search is performed in the problem space where each state is defined by a problem topology and is represented as a constraint graph. CHS augments the definition of a problem space by refining a state to include problem topology (structure), textures (measures of structure) and objective (rating alternative solutions) [Fox, 1989].

Related Work

In a prototype system called CRIPS, Liu demonstrated how constraint and rule-based reasoning can be integrated into a hybrid problem-solving paradigm to deal with the problem of representing disjunctions in rule-based systems [Liu, 1994]. Specifically, the disjunctions are represented as constraint variables and their domains and the relations among disjunctions are represented as constraints. Our work extends this effort by integrating a more general form of constraint based reasoning into an integrated rule-based cognitive architecture.

A hybrid learning system was developed for automating the acquisition of problem-solving knowledge using CSP approaches [Subramanian, Freuder, 1990]. The technique proposed compiles production rules from the observation of constraint-based problem solving behaviour. The compiled rules corresponded to the values deleted by the constraint-based problem solving. In contrast to the work of Subramanian and Freuder, our research proposes to utilize Soar's learning ability to encode texture-based problem independent heuristics as learned rules.

The Adaptive Constraint Engine (ACE) is described as "cognitively oriented" architecture that can learn from experience solving problems in the CSP domain [Epstein, et. al., 2002]. ACE is built upon an underlying general problem solving and learning architecture called FORR (FOr the Right Reasons) which represents knowledge as a three-tiered hierarchy of procedures or "advisors."

Varieties of learning within Soar have been extensively investigated, including across different tasks, and the types it has not yet exhibited include: vary large problems; complex analogies; similarity-based generalization; and, representational shifts [Steier, et. al, 1987]. Our work seeks to reason over a much more abstracted problem space thereby producing more generalized chunks applicable to a much broader range of problem domains.

Design of CHS-Soar

There are two central, yet inter-related design goals of CHS-Soar. First is the use of domain independent problem solving techniques and the second is domain independent learning. Specifically, CHS introduces a standard CSP problem representation which encompasses the use of variables, which have a domain of discrete values, and constraints between variables. CHS also allows us to introduce general purpose problem solving techniques such as propagation and the use of domain independent problem structure textures to guide variable and value selection. The second design goal is to focus Soar's internal reasoning on the selection of textures measures to facilitate the learning of domain independent problem solving rules ideally useful across different problem types. Architecturally (Figure 1) CHS-Soar is composed of three parts: [1] the Soar kernel; [2] Soar agent (production rules); and, [3] the external agent. A more detailed overview of the design of CHS-Soar is provided by Bittle and Fox [Bittle, Fox, 2008].

Soar Kernel

The Soar kernel is invariant and encompasses a fixed set of computational mechanisms including: working memory (state representation); long term memory (repository for production rules); decision cycle (used to link working and long term memory); and, learning system.

Soar Agent (Production Rules)

Production rules provide the "programming language" for developers to create different Soar programs, called agents, and form the content of Soar's Long Term Memory. Soar's general problem-solving ability results from the fact that the same architecture (the Soar kernel) can be used to solve different problems as defined by different sets of production rules. Rules are used to; propose, select and apply operators.

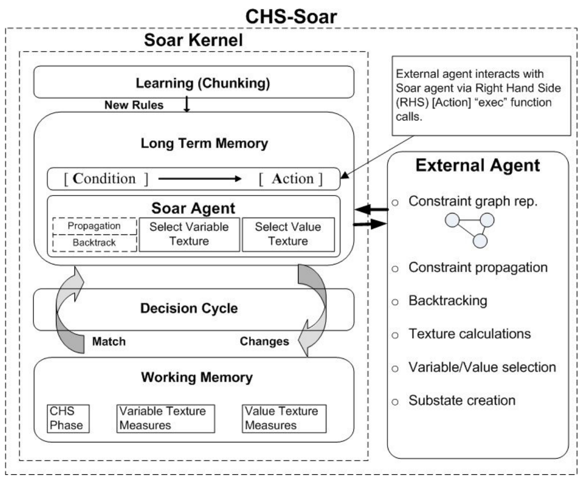

Figure 1: CHS-Soar Architecture

Operators perform actions and are consequently the locus of decision making [Laird et. al, 1987]. CHS-Soar operators are focused on variable and value texture measure selection. CHS-Soar operators provide the framework to introduce the CHS problem solving cycle.

CHS-Soar Problem Solving Cycle Soar's three-phase decision cycle (proposal, decision, and apply) provides a suitable framework to introduce the CHS problem solving cycle summarized in Table 1.

Table 1: CHS Problem Solving Cycle

```
Repeat
    1   Conduct propagation (or backtrack) within state
          • calculate variable texture measures
    2   Select variable (based on heuristics as suggested by
        textures measures)
          • calculate value texture measures
    3   Select value (based on heuristics as suggested by
        textures measures)
Until (all variables values instantiated & all constraints satisfied)
```

As depicted in Figure 2, three Soar decision cycles are performed during one CHS cycle. Instrumental to the learning of domain independent problem-solving rules is the requirement to have the Soar agent component of the CHS cycle only reason about the selection of normalized variable and value textures measures — not actual problem variable and values. Consequently the Soar agent is decoupled from the actual problem constraint graph. The Soar agent interacts with the external agent via production action-side function calls.

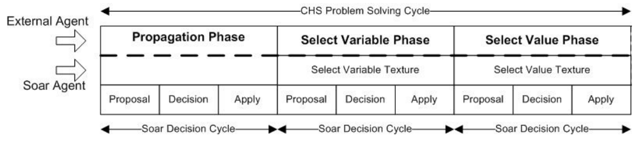

Figure 2: CHS-Soar Problem Solving Cycle

External Agent

The external agent is an optional user-defined program where Soar agent production rule functions reside. From the external agent we can register new functions which can extend the functionality of Soar agent productions. The CHS external agent provides five functions including: CSP problem representation (binary constraint graph); constraint propagation using the AC-3 algorithm [Mackworth, 1977]; chronological backtracking; and, variable and value texture calculations.

Texture Measures Textures are structural measures of the constraint graph used to guide the order of variable and value selection [Fox et. al., 1989]. In order to demonstrate the future introduction of more insightful texture measures [Fox, 1990], three frequently cited [Kumar, 1992] variable and value ordering measures (and their associated heuristics) are utilized as outlined in Table 2.

Table 2: Texture Measures and Associated Heuristics

Name	Texture Measures	Heuristics
Minimum Remaining Value (MRV)	D_i, Number of remaining values in domain.	Select the variable with the smallest D_i, value
Degree (DEG)	C_i, number of constraints linked to variable.	Select the variable with the largest C_i value
Least Constraining Value (LCV)	F_i, number of available values in domain of linked variables not instantiated.	Select the value with the largest F_i value

Of particular note is the separation of the texture measure from its associated unary heuristic. In order to facilitate learning across different problem sizes and more importantly different problem domains, texture measures are normalized between 0 (minimum value) and 1 (maximum value). A "pruned" (duplicate values removed) list of normalized texture measures are returned to the Soar agent. Pruning has the effect of further decoupling the texture measures from any specific problem domain.

Soar and External Agent State Representation

The Soar agent initially establishes a binary constraint graph in Soar's working memory. Once problem solving begins, the Soar agent state representation is defined by the CHS phase (i.e. select variable), state status, desired state, and the collection of normalized texture measures. In contrast, the external agent maintains a complete binary constraint graph problem representation.

Subgoaling and Learning

Planning in Soar arises out of its impasse detection and substate creation mechanism. Soar automatically creates a subgoal whenever the preferences are insufficient for the decision procedure to select an operator [Laird, et. al, 1987]. Soar includes a set of default production rules that allow subgoaling to be performed using a simple look-ahead search which CHS-Soar utilizes to evaluate variable and value texture measure performance. After propagation, the Soar agent has no prior knowledge of which variable and/or value texture measure to select (assuming no previous learning). Soar will detect an impasse and automatically subgoal to evaluate each texture measure returned and now cast as operators.

Texture Measure Evaluation CHS-Soar uses three separate, yet related texture measure evaluations. The first is a simple numerical evaluation which gives a higher preference to the texture measure that requires fewer propagation cycles to achieve a candidate solution. Second, if a texture measure leads to a substate domain blow-out it is symbolically evaluated as a failure and rejected. Third, if we cannot match a texture measure in the substate (i.e. an updated constraint graph is generated that does not include the texture measure under evaluation) the texture measure is symbolically evaluated as a "partial failure."

Since CHS requires both variable and value selections to advance a solution, in order to facilitate the evaluation of any substate variable texture measure we are required to make some type of value texture measure commitment leading to a variable – value texture measure pairing. The approach utilized is to allow Soar to further subgoal (i.e. a sub-subgoal) about value texture selection based on the value texture measures generated for each variable texture measure under consideration. Within each sub-substate, the variable — value pairing is held constant and a "candidate" solution is attempted.

The second design goal of CHS-Soar is the learning of domain independent problem solving rules — specifically rules that can be transferred and effectively used on different problem types. Chunks are created in the CHS-Soar agent as we resolve impasses. Resulting chunks have as their conditions either unary or binary conditions composed of texture measures (cast as operators).

Implementation of CHS-Soar

The CHS-Soar agent was developed using Soar version 8.6.3 (Windows versions). The external agent was developed in C++ using Windows Visual Studio 2005.

Experiments

The paper reports on two selected computational experiments performed to-date. The objective of experiment 1 is to establish the subgoaling, learning and transfer of learning ability of CHS-Soar. Experiment 2 considered the impact of problem size/complexity on the performance of externally learned chunks. Problem instances were run 10 times and averaged. Experiments were conducted for two sample problems: [1] map colouring (MC); [2] job-shop scheduling (JSS), both well known combinatorial problems. The relationship of the map colouring problem and the more formal four-colour problem to general intelligence is highlighted by Swart [Swart, 1980] who notes one of the main attractions of the problem lies in the fact that it is "so simple to state that a child can understand it." The JSS problem instance is taken from [Pang, et. al, 2000]. Results are presented in terms of "Decisions (cycles)" — the basic unit of reasoning effort in Soar (see Figure 2).

Experiment 1: Learning and Transfer of Learning

The first experiment investigated the impact of subgoaling, internal learning and the transfer of externally learned rules between problem types. In order to assess the performance of an external set of learned chunks on the map colouring problem, the chunks acquired through internal learning from the JSS problem were introduced for test case 4 (see Table 3) and vise versa for the JSS.

Table 3: Learning and Transfer of Learning Test Cases

Ref	Test Case	Description
1	Random	Random selection of variable and value texture measures (indifferent).
2	Hard-Coded	Explicitly coded MRV, DEG variable selection heuristics and LCV value selection heuristic.
3	Internal	Problem is run using internally acquired chunks.
4	External	Problem is run using externally acquired chunks from different problem domain.

Results and Discussion

Figure 3 presents the number of decisions required to solve the CHS-Soar model of a map colouring problem (11 variables and 23 constraints) for the test cases outlined in Table 3. As highlighted the hard-coded variable and value texture selection heuristics (case 2) result in a lower number of decisions (41% reduction) then when using a random selection of texture measures (case 1). When subgoaling and learning were enabled (case not shown), CHS-Soar generated on average 60 chunks. Case 3 demonstrates improved problem solving using the 60 internally acquired learned rules as compared to the explicit use of hard-coded heuristics (case 2) resulting in a 21% reduction in decision cycles. For case 3 it was observed that CHS-Soar used both a combination and range of variable and value texture measure types and values to secure a solution.

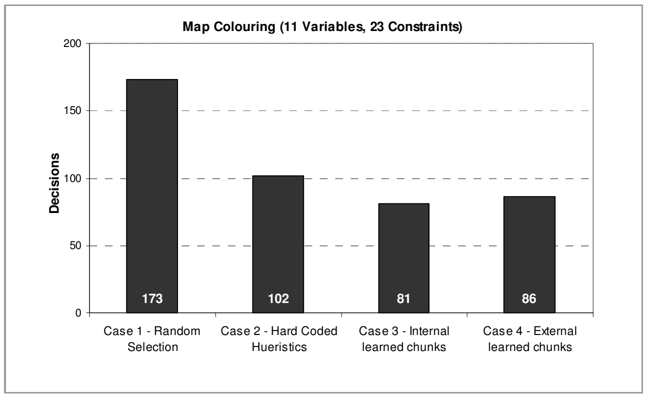

Figure 3: Map Colouring-Decisions as a Function of Subgoaling/Learning Test Cases.

Case 4 demonstrates the domain independent problem solving performance of CHS-Soar chunks using 177 chunks externally learned from the JJS problem. Case 4 resulted in a 16% reduction in decision cycles over case 2 (hard-coded heuristics). Yet for case 4 we observe a 6% increase in decision cycles as compared case 3 (internal learned chunks).

Figure 4 presents the number of decision cycles required to solve the CHS-Soar model of a 3x5 Job-Shop Schedule (JSS) problem (15 variables and 35 constraints) for the test cases outlined in Table 3. Similar to the map colouring problem instance, we can observe a reduction in decision cycles (29% lower) using hard coded heuristics (case 2) over the random selection (case 1). Subgoaling and learning (one training run) resulted in 177 internally learned chunks. We note a 23% reduction in decision cycles when the problem is solved using internally learned rules (case 3) as compared with the explicit use of hard coded heuristics (case 2). Case 4 illustrates the impact of using 60 externally learned rules acquired from the map colouring problem. Case 4 shows a 10% reduction in the number of decision cycles as compared to case 2 (hard coded).

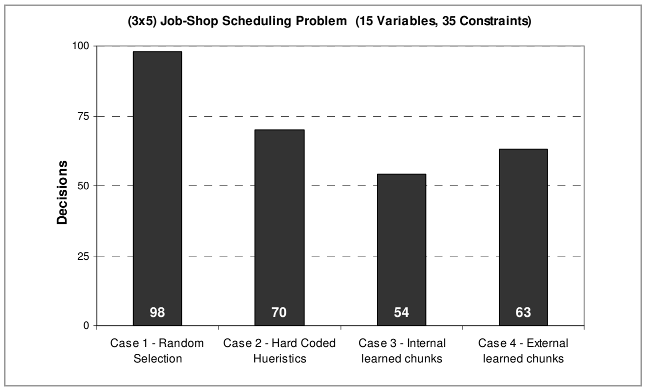

Figure 4: JSS-Decisions as a Function of Subgoaling/Learning Test Cases.

Experiment 2: Scalability

The second experiment explored the performance of externally learned chunks as a function of problem complexity for the map colouring problem. Specifically, the 177 learned chunks from the job-shop scheduling problem (3x5) were introduced into progressively larger map colouring problems ranging in size from 7 variables (9 constraints) to 44 variables (103 constraints).

Results and Discussion

Figure 5 presents a comparison of decisions cycles for 3 selected test cases (see Table 3 as a function of problem complexity for the map colouring problem. For case 5, the 177 externally learned chunks from the 3x5 JJS problem were introduced to solve each MC problem instance.

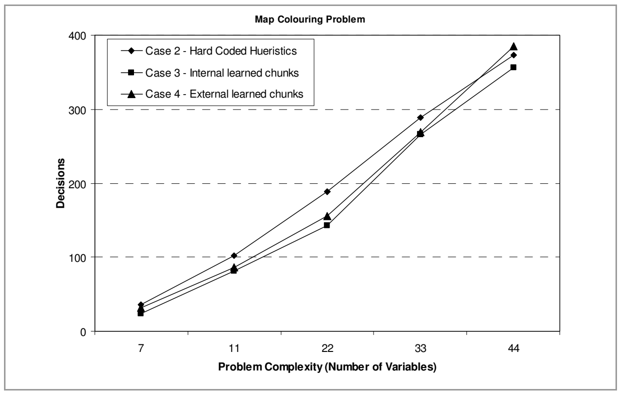

Figure 5: Map Colouring-Comparison of Decisions versus Problem Complexity for Selected Test Cases.

As illustrated for this specific problem series, the externally acquired learned rules demonstrate similar problem solving performance, to both the hard-coded heuristics and internally generated chunks as problem complexity increases.

Conclusions and Future Work

While preliminary, work to date has demonstrated how a specific form of constraint-based reasoning (CHS) can be effectively introduced into a symbolic cognitive architecture (Soar) using a generalized set of 34 productions. This results in an integration of two important types of reasoning techniques, namely constraint propagation and rule chaining.

We have demonstrated the ability of CHS-Soar to learn rules while solving one problem type (i.e., graph colouring) that can be successfully applied in solving another problem type (i.e., Job Shop Scheduling). CHS and specifically the use of texture measures allow us to transform a problem specific search space (based on variables and values) into a more generalized one based on abstracted texture measures which provides the environment to achieve domain independent learning.

Future work will include an expanded portfolio of test case problems to further validate and better understand CHS-Soar ability to learn and use domain independent texture based rules for more practical problems. More insightful texture measures, support augmentation (i.e. frequency) and improved texture evaluation functions are also being investigated as well as a texture based "discovery system."

References

Bittle, S.A., Fox, M.S., (2008), "Introducing Constrained Heuristic Search to the Soar Cognitive Architecture", Technical Report, Enterprise Integration Laboratory http://www.eil.utoronto.ca/other/papers/index.html.

Diaper, D., Huyck, C., Amavasai, B., Cheng, X. and Oussalah, M. 2007. Intelligently Engineering Artificial Intelligence Engineering: The Cognitive Architectures Competition. Proceedings of the workshop W3 on Evaluating Architectures for Intelligence, at the Association for the Advancement of Artificial Intelligence annual conference (Vancouver).

Epstein, S. L., Freuder, E.C., Wallace, R.J, Morozov, A., Samuels, B. 2002. The Adaptive Constraint Engine. In P. Van Hentenryck, editor, Principles and Practice of Constraint Programming -- CP 2002: 8th International Conference, Proceedings, volume LNCS 2470 of Lecture Notes in Computer Science, pages 525--540. SpringerVerlag, 2002.

Fox, M.S., Sadeh, N., Bayken, C., 1989. Constrained Heuristic Search. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, pg:309– 315.

Fox, M.S., 1986. Observations on the Role of Constraints in Problem Solving, Proceedings Sixth Canadian Conference on Artificial Intelligence, Montreal, Quebec.

Kumar, V., 1992. Algorithms for constraint-satisfaction problems: A survey. AI Magazine, 13(1):32-44, 1992

Laird, J.E., Newell, A., Rosenbloom, P. 1987. Soar: An Architecture for General Intelligence. Artificial Intelligence, 33: 1-64.

Laird, J. E. 2008. Extending the Soar Cognitive Architecture. Artificial General Intelligence Conference, Memphis, TN.

Langley, P., Laird, J., Rogers, S., 2006. Cognitive Architectures: Research Issues and Challenges, Technical Report, Computational Learning Laboratory, CSLI, Stanford University, CA..

Liu, B., 1994. Integrating Rules and Constraints Proceedings of the 6th IEEE International Conference on Tools with Artificial Intelligence (TAI-94), November 6-9, 1994, New Orleans, United States, 1994.

Mackworth, A.K., 1977. Consistency in Networks of Relations, J. Artificial Intelligence, vol. 8, no. 1, pp. 99-118, 1977.

Pang, W., Goodwin, S.D., 2004. Application of CSP Algorithms to Job Shop Scheduling Problems, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.9563.

Shivers, O., 1986. Constraint propagation and macro-compilation in Soar. In Proceedings of the Soar Fall '86 Workshop, November 22, 1986.

Steier, D.M., Newell, A., Flynn, R., Polk, T.A., Unruh, A., 1987. "Varieties of Learning in Soar." The Soar Papers: Research on Integrated Intelligence, Volume 1, Chapter 18, Pages 536-548, MIT Press, Cambridge, Massachusetts.

Subramanian, S.; Freuder, E.C. 1990. Rule compilation from constraint-based problem solving. Tools for Artificial Intelligence, 1990, Proceedings of the 2nd International IEEE Conference on, Vol., Iss, 6-9 Nov 1990, pp: 38-47.

Swart, E.R., 1980. "The philosophical implications of the four-color problem". American Mathematical Monthly (JSTOR) 87 (9): 697--702. http://www.joma.org/images/upload_library/22 /Ford/Swart697-707.pdf.

Wallace, M., 1996. Practical applications of constraint programming, Constraints, An International Journal, 1, 139-168.