Learning and Using Hyper-Heuristics for Variable and Value Ordering in Constraint Satisfaction Problems

Sean A. Bittle
Department of Mechanical and Industrial Engineering,
University of Toronto
King's College Road
Toronto, Ontario, Canada, M5S 3G9
416-978-682
sean.bittle@utoronto.ca

Mark S. Fox
Department of Mechanical and Industrial Engineering,
University of Toronto
King's College Road
Toronto, Ontario, Canada, M5S 3G9
416-978-682
msf@eil.utoronto.ca

ABSTRACT
This paper explores the use of hyper-heuristics for variable and value ordering in binary Constraint Satisfaction Problems (CSP). Specifically, we describe the use of a symbolic cognitive architecture, augmented with constraint based reasoning as the hyper-heuristic machine learning framework. The underlying design motivation of our approach is to "do more with less." Specifically, the approach seeks to minimize the number of low level heuristics encoded yet dramatically expand the expressiveness of the hyper-heuristic by encoding the constituent measures of each heuristic, thereby providing more opportunities to achieve improved solutions. Further, the use of a symbolic cognitive architecture allows us to encode hierarchical preferences which extend the effectiveness of the hyper-heuristic across problem types. Empirical experiments are conducted to generate and test hyper-heuristics for two benchmark CSP problem types: Map Coloring; and, Job Shop Scheduling. Results suggest that the hyper-heuristic approach provides a dramatically higher level of representational granularity allowing superior intra-problem and inter-problem solutions to be secured over traditional combinations of variable and value ordering heuristics.

Categories and Subject Descriptors
I.2.8 Problem Solving, Control Methods, and Search

General Terms
Algorithms

1. INTRODUCTION
In this paper we present a novel approach for learning and using hyper-heuristics to address the problem of variable and value ordering in binary constraint satisfaction problems. Our approach uses a symbolic cognitive architecture, augmented with constraint based reasoning as the hyper-heuristic machine learning framework. Research has demonstrated variable and value ordering heuristics can dramatically improve the efficiency of search [7]. While a variety of heuristics have been developed to guide both variable and value ordering, most are problem specific; experimentation is often required to select heuristics for new problem types and few have demonstrated their effectiveness across a range of problem types [4, 13]. Consequently, the central problem with using heuristics to guide variable and value ordering is the tradeoff between generality and effectiveness.

One recent technique to achieve more effective generality is the use of hyper-heuristics or *heuristics to choose heuristics*. More formally, a hyper-heuristic is a high-level heuristic which uses some type of learning mechanism in order to choose between various low-level heuristics [2]. Often hyper-heuristics achieve some level of generality at the expense of computational effort by encoding a large yet fixed number of low level heuristics.

The machine learning approach used in the current study is CHS-Soar [1]. CHS-Soar integrates two different problem-solving paradigms — constraint and rule-based reasoning — into a unified, declarative symbolic architecture. This is achieved by introducing Constrained Heuristic Search (CHS) [6] to the Soar cognitive architecture [8]. The approach allows us to encode the relative performance of all constituent measures associated with each low level heuristic and thereby expands the number of selection operators beyond the fixed number of low level heuristics considered.

2. RELATED WORK
Historically, the principal application of hyper-heuristics has been in the area of optimization where the hyper-heuristic is focused on selecting between a set of low level meta-heuristics. For example a unified graph based hyper-heuristic (GHH) framework is defined in which different local search based algorithms are considered to search upon sequences of low level graph coloring heuristics for university timetabling problems [10].

The idea of using hyper-heuristics for variable ordering in binary constraint satisfaction problems has also been considered by Terashima-Marin et.al.[14]. Their overall approach is based on using a variable-length Genetic Algorithm (GA) where the chromosome is made up of a series of blocks representing condition-action rules used to switch between seven different variable ordering heuristics.

The Adaptive Constraint Engine (ACE) is described as a "cognitively oriented" architecture that can learn from experience in solving problems in the CSP domain [3]. ACE's learning mechanism is focused on learning the appropriate importance, called "weights", of individual advisors. ACE manages search heuristics and attempts to learn new heuristics for variable and value ordering. It then extrapolates what it learns on simple problems to solve more difficult ones. Because ACE does unsupervised learning through trial and error with delayed rewards, it qualifies as a reinforcement learner.

3. SOLUTION APPROACH: CHS-SOAR

CHS-Soar problem solving is formulated by applying operators to states within a problem space in order to achieve a goal. A goal or solution in CHS-Soar is a complete assignment of variables which satisfy all problem constraints. A state in CHS-Soar is represented as a binary constraint graph. Each change to the constraint graph ─ initiated by a new variable and/or value assignment ─ corresponds to a new state. CHS-Soar operators are used to transform states via variable and value ordering decisions. Consequently, problem solving in CHS-Soar can be described as the process of moving state to state in a problem space by heuristically choosing variables and values to instantiate. This allows us to incrementally move from an initial to a final goal state. Finally, we consider an "improved" or "superior" CSP solution to be one that requires a lower total number of consistency checks to secure a solution.

3.1 Textures and Heuristics Considered

Textures describe structural features of the constraint graph [6]. A heuristic usually relies on an associated set of textures to make a decision; consequently, textures can be viewed as the constituent parts of a heuristic. For example, consider the well known Minimum Remaining Value (MRV) heuristic which is often cited for variable ordering in CSPs [7]. It is useful to differentiate between the MRV heuristic and the MRV textures. The MRV heuristic suggests we select the variable that has the smallest domain of legal values remaining. In order to use this heuristic we first need to tabulate the MRV textures, namely the number of remaining values in each variable domain. While the MRV heuristic is focused on the variable with the smallest texture value, we are still required to calculate all MRV textures to locate the smallest one. In the current work we are interested in reasoning about the impact of each individually observed texture value.

In order to demonstrate the efficacy of our hyper-heuristic approach we consider two frequently [7] cited variable ordering heuristics, e.g. MRV and DEG, and one value ordering heuristic, e.g. LCV, and associated textures as outlined in Table 1.

Table 1. Textures and Associated Ordering Heuristics

Name	Texture	Heuristic
Minimum Remaining Values (MRV)	D_i, number of remaining values in domain.	Select the variable with the smallest D_i, value i.e. pick the variable with the fewest legal values.
Degree Heuristic (DEG)	C_i, number of constraints linked to variable.	Select the variable with the largest C_i value i.e. pick the variable that is involved in the largest number of constraints on other unassigned variables.
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 i.e. pick the value that rules out the fewest choices for the neighboring variables in the constraint graph.

3.2 Hyper-Heuristic Representation

Central to the design of an effectiveness hyper-heuristic is a suitable heuristic ordering mechanism or switch function [2]. The switch essentially decides, based on selected state measures (which we call textures), when to select and apply each encoded heuristic. A variety of techniques have been investigated to encode the hyper-heuristic switch algorithm with the most popular approach based on some form of Genetic Algorithm (GA) encoding [11]. GA approaches are "integrative" in that they require a training phase comprising a large number of training instances to generate an averaged set of conditions which trigger a discrete heuristic action. It is useful to note that while GA approaches generate very compact hyper-heuristic representations, they achieve compactness at the expense of discarding almost all of the rich structural "detail" generated. The structural detail refers to the underlying textures associated with each heuristic. Further, the level of expressiveness for each switch "action" is typically the selection of a specific heuristic. Consequently the effective generality of the hyper-heuristic is limited by the number of heuristics encoded as possible switch actions.

The design of our hyper-heuristic seeks to capture and exploit the influence of this underlying texture detail. We do this by learning the problem-solving impact of each individual texture value. Our hyper-heuristic operators are defined by a tuple of four data elements: texture type, e.g. MRV; texture value, e.g. MRV = 0.3; relative frequency; and percent problem complete, e.g. tuple ≡ <type, value, relative frequency, percentage complete>. In order to generalize our hyper-heuristic; for each texture type, e.g. MRV, DEG, LCV, we prune out duplicate texture type values and normalize each value between 0 (minimum value) and 1 (maximum value).

3.3 Hyper-Heuristic Generation

The hyper-heuristic generation process employs Soar's internal reasoning ability called "subgoaling." Internal reasoning in Soar arises out of its impasse detection and substate creation mechanism. CHS-Soar operators are used to transform states via variable and value ordering decisions. CHS-Soar automatically creates a subgoal whenever preferences are insufficient for the decision procedure to select a variable and/or value ordering operator.

Our hyper-heuristics are generated as a by-product of problem-solving. After propagation, CHS-Soar generates an updated set of variable textures, e.g. MRV and DEG, and the associated set of tuples, e.g. tuple ≡ <type, value, relative frequency, percentage complete>. CHS-Soar then dynamically casts each tuple as an operator. Since CHS-Soar has no prior knowledge as to which operator to select (assuming no previous learning) it will detect an impasse and automatically subgoal to evaluate each operator in order to establish a clear preference for one. A separate problem space is created allowing it to perform a simple look-ahead search in order evaluate the problem solving impact of each operator. The evaluation of each operator involves assigning a symbolic and/or numeric preference to each operator currently under evaluation. The numerical evaluation assigns a higher preference to an operator that can advance a current partial solution the farthest using the fewest number of consistency checks. Once CHS-Soar has resolved which variable, and then, which value ordering operator to select, it encodes, i.e. learns, the results.

Soar includes a single, uniform learning mechanism, called "chunking", that converts the results of problem solving in subgoals into new production rules called "chunks." While encoded using Soar syntax, chunks have a Condition → Action semantic structure. Resulting chunks have either a unary or binary condition composed of operators. Chunk actions encode the symbolic and/or numeric preferences for these operators. Texture values are normalized between 0 and 1 to extend their generality. Figure 1 provides a pseudo-code example of a binary chunk. At a macro level, the generated "hyper-heuristic" is the aggregate collection of a large number, perhaps thousands, of individual chunks, i.e. production rules.

```
Condition [Op1 ≡ (MRV = 0.2) and Op2 ≡ (DEG
= 1.0)] → Action [(Op1) Preference (Op2)]
```

Figure 1: Pseudo-code example of a binary chunk.

4. EXPERIMENTS

We report on two experiments conducted that assess the intra, i.e. within, and inter, i.e. across, problem type problem solving characteristics of the hyper-heuristic design approach. We consider two benchmark problem types: Map Coloring Problem (MCP); and, the Job Shop Scheduling Problem (JSP). The MCP instances were generated and adapted from [5]. The JSP instances were generated and adapted from [12]. For each problem type, e.g. Map Coloring, we have a set of problem instances. Each problem instance corresponds to a different size of that problem type. Results are presented in terms of "consistency checks." As noted by [14] the count of consistency checks is a common criterion to measure the efficiency of a CSP search algorithm. CHS-Soar utilizes the AC-3 algorithm [9] to conduct propagation and, when required, employs simple chronological backtracking. In all experiments the benchmark for comparison uses standard CSP techniques where the variable ordering heuristic is the MRV heuristic, i.e. select variable with minimum remaining values in domain. If the MRV cannot select a variable we use the DEG heuristic, e.g. select variable that participates in the most constraints on unassigned variables. For value ordering we use the Least Constraining Value (LCV) heuristic which selects the domain value that is least constraining to other unassigned variables. If the LCV heuristic does not identify a unique value, the value is selected randomly. Each problem instance was solved 10 times and results averaged.

4.1 Experiment 1

Experiment 1 tests the hypothesis that expanding the expressiveness of the hyper-heuristic beyond the simple encoding of each low level heuristic operator will secure superior intra-problem type solutions over traditional combinations of fixed unary variable and value ordering heuristics. Specifically, for each problem type and size instance considered, we encode a "custom" hyper-heuristic. Figures 2 and 3 compare the performance of the custom hyper-heuristics to the benchmark for each problem type. As illustrated for the MCP and JSP problem types we can observe that each custom hyper-heuristic secures improved problem solving performance over the benchmark for each problem size instance considered. Further, we can see that as problem size increases for the MCP and JSP problem types we can observe a relative increase in the overall problem solving performance of the custom hyper-heuristics as compared to the results delivered by the benchmark combination of heuristics. Results suggest for these problem types, as problem size increases more opportunities for improved solutions are observed and exploited by the hyper-heuristic.

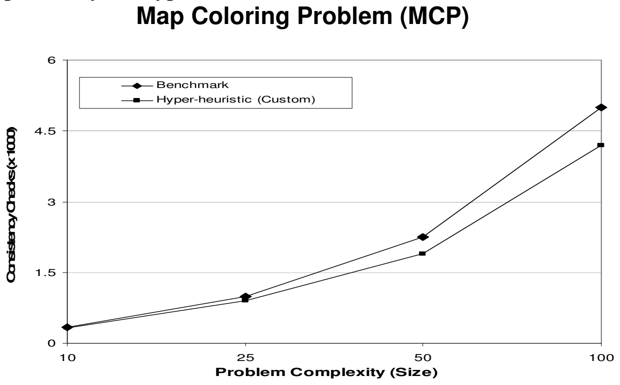

Figure 2. Custom Hyper-Heuristic Performance as a Function of Problem Complexity for the Map Coloring Problem.

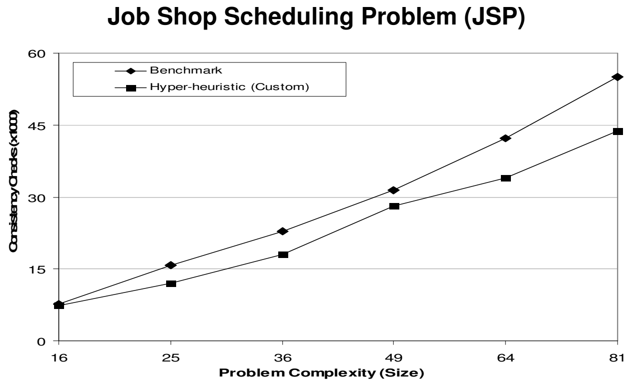

Figure 3. Hyper-Heuristic Performance as a Function of Problem Complexity for the Job Shop Scheduling Problem.

4.2 Experiment 2

Experiment 2 explores the inter-problem solving effectiveness of hyper-heuristics generated from one problem type and used to solve a different problem type. Specifically, for each problem type, we generate a single hyper-heuristic from the smallest problem instance considered. We then transfer and use it to use solve a different problem type.

Figures 4 and 5 compare the performance of the hyper-heuristics generated for each problem type and used to solve a different problem type. As illustrated for each problem type, we can observe that the transferred hyper-heuristics secure improved problem solving performance over the benchmark for the range of problem sizes considered. However, we note that neither of the transferred hyper-heuristics secures superior problem solving performance to the custom (Experiment 1) hyper-heuristics. These results would appear to support the observation, that as the hyper-heuristics generated from one problem type are applied to problems that are different in size as well and type, their effective generality appears to diminish incrementally as problem complexity increases.

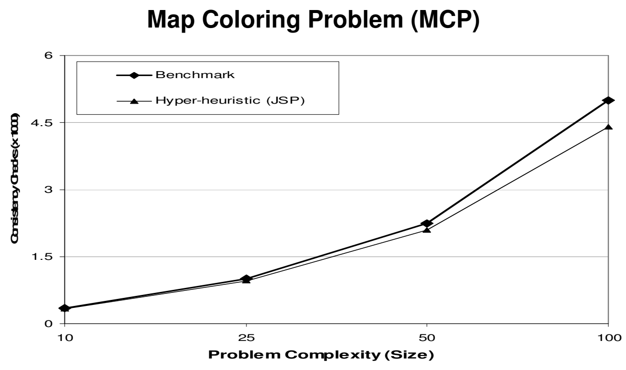

Figure 4. JSP Hyper-Heuristic Performance as a Function of Problem Complexity for the Map Coloring Problem.

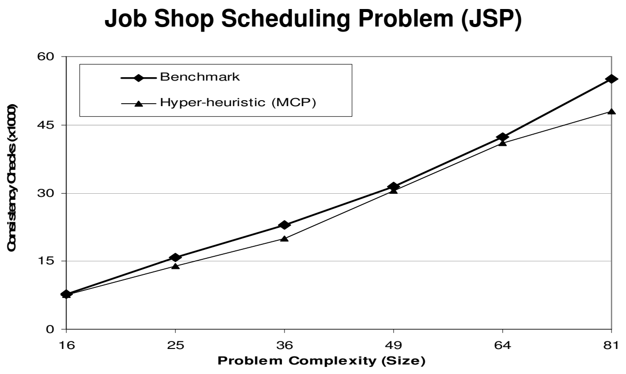

Figure 5. MCP Hyper-Heuristic Performance as a Function of Problem Complexity for the Job Shop Scheduling Problem.

5. CONCLUSIONS

This paper has demonstrated how hyper-heuristics can be generated and used for variable and value ordering in binary Constraint Satisfaction Problems (CSP). We have outlined a novel technique to encode hyper-heuristics using a symbolic cognitive architecture (Soar), augmented with constraint based reasoning (CHS) as the machine learning framework. Results suggest this approach confers a number of observed design benefits over traditional hyper-heuristic representations and evolutionary encoding approaches.

First, using a minimum number of low level heuristics, our approach dramatically expands the expressiveness of the hyper-heuristic by encoding the constituent textures of each heuristic — not simply the low level heuristics. Second, the approach encodes and exploits the rich problem solving "detail" associated with the possible selection of each constituent texture. This is in contrast to evolutionary approaches which must integrate and average training insight. Finally, the use of a symbolic cognitive architecture allows us to encode hierarchical preferences which extend the effectiveness of the hyper-heuristic across problem types.

We have demonstrated the ability to discover, learn and use texture based hyper-heuristics for variable and value ordering that produce superior intra-problem solving performance over traditional combinations of unary heuristics for two problem types. Further, we have demonstrated the ability to learn a hyper-heuristic while solving one problem type can be successfully applied in solving a different problem type and deliver superior problem-solving performance over traditional combinations of unary heuristics.

6. REFERENCES

[1] Bittle, S.A., Fox, M.S. Introducing Constrained Heuristic Search to the Soar Cognitive Architecture, *Second Annual Conference on Artificial General Intelligence*, Arlington Virginia. 2009.

[2] Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P. Schulenburg, S. Hyper-heuristics: an emerging direction in modern search technology, *Handbook of Metaheuristics*, chapter 16, Hyper-heuristics: an emerging direction in modern search technology, pp. 457--474. Kluwer Academic Publishers, 2003.

[3] Epstein, S. L., Freuder, E.C., Wallace, R.J, Morozov, A., Samuels, B. 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.

[4] Gent., I.P., MacIntyre, E., Prosser, P., Smith, B.M., Walsh, T. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem, *Principles and Practice of Constraint Programming- CP'96*, pp 179–193, 1996.

[5] Fleurent C. Ferland J.A. Genetic and hybrid algorithms for graph coloring, Annals of Opns Res. 63, 437-461. 1996.

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

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

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

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

[10] Qu, R., Burke, E.K. Hybridisations within a Graph Based Hyper-heuristic Framework for University Timetabling Problems. *to appear at Journal of Operational Research Society (JORS), 2008.* Online publication Oct, 2008. doi: 10.1057/jors.2008.102

[11] Soubeiga, E. Development and application of hyperheuristics to personnel scheduling. *PhD Thesis, University of Nottingham.* 2003.

[12] Taillard, E. Benchmarks for basic scheduling problems, European Journal of Operational Research 64, (1993), 278-285

[13] Tsang, E. Foundations of Constraint Satisfaction. *Academic Press*. 1993.

[14] Terashima-Marín, H., Ortiz-Bayliss, J.C., Ross, P., Valenzuela-Rendón, M. Using Hyper-heuristics for the Dynamic Variable Ordering in Binary Constraint Satisfaction Problems. *MICAI 2008*: 407-417 2008.