A Hyperheuristic Approach to Select Enumeration Strategies in Constraint Programming

BRODERICK CRAWFORD
MAURICIO MONTECINOS
Pontificia Universidad Católica de Valparaíso
Valparaíso, CHILE
firstname.name@ucv.cl

CARLOS CASTRO
Universidad Técnica Federico Santa María
Valparaíso, CHILE
firstname.name@utfsm.cl

ERIC MONFROY
Université de Nantes
Nantes, FRANCE
firstname.name@univ-nantes.fr

Abstract: In this work we exploit search process features to dynamically adapt a Constraint Programming solver in order to more efficiently solve Constraint Satisfaction Problems. Our proposal uses a hyperheuristic to decide adaptation possibilities. The main novelty of our approach is that we reconfigure the search based solely on performance data gathered while solving the current problem. We report encouraging results where our combination of strategies outperforms the use of individual strategies.

Key–Words: Constraint Programming, Enumeration Strategies, Variable Ordering Heuristics, Value Ordering Heuristics, Choice Function

1 Introduction

Constraint Programming (CP) has been defined as a technology of Software used to describe and solve combinatorial problems. The main idea of this paradigm is to model a problem by mean of a declaration of variables and constraints and to find solutions that satisfy all the constraints. Many of the combinatorial problems focused by CP can be modeled like a Constraint Satisfaction Problem (CSP), which consist of a sequence of variables $X = x_1, x_2, ..., x_n$ with its respective domains $D = D_{x_1}, D_{x_2}, ..., D_{x_n}$, and a finite set $C$ of constraints restricting the values that the variables can take simultaneously [14]. The goal is to assign a value to each variable satisfying all the constraints. The most general notation for CSP is the following [16]: $< C; x_1 \in D_{x_1},..., x_n \in D_{x_n} >$.

The basic mechanism underlying CP to solve a CSP interleaves Constraint Propagation (network consistency) and Enumeration (distribution or labeling) [1]. In essence, the algorithm increases the efficiency of the search by looking ahead actively using the constraints to prune the search space. Furthermore, an optimization approach is feasible from constraint satisfaction in a form of branch and bound. That is, as soon as a solution is found, a further constraint is added forcing future values to be better than the value just found according to the optimization criteria. This causes the system to backtrack until a better solution is found. When no further solutions can be found the optimum value is reached. A CP structure is shown in Figure 1.

This work is focused in the Enumeration phase of CP, where the use of variable and value selection heuristics is critical. A suitable definition and use of the enumeration strategy can improve the resolution process strongly.

However, the effect of strategies is generally unpredictable. We propose to dynamically change strategies showing bad performance. Our proposal uses a hyperheuristics to decide which method is best for a given step. This approach would improve the applicability of CP to different types of problems without the fine tuning of the solver.

We apply to solve N-Queens and Magic Square benchmark problems, different variables and values ordering heuristics presented in the literature [12, 3] in comparison with our proposal.

This paper is organized as follows. In Section 2 is described adaptive solving problems. Section 3 and 4 show the resolution technique and the enumeration strategies. In Section 5 is described how measuring the search. Section 6 described how controlling CP search with a hyperheuristic. We present experimental results obtained in Section 7. Finally, conclusions are presented in section 8.

2 Adaptive Solving Problems

The concept of adaptation, is the ability of a solver to change its behaviour when is confronted to a problem that it is very difficult or impossible to resolve with the configuration used. In problem solving we can found
many examples of adaptation.

In Adaptive Constraint Engine (ACE) [8] adaptation is achieved mixing learning with heuristic, through a learning architecture based on use of multiple heuristics. All this is supported by a series of procedures called advisors, where each represents a general principle that supports expert behavior. ACE is equipped with Variable Selection Heuristics such as: maximum domain size, minimum domain size, maximum degree y minimum degree. These heuristics are embedded in procedures called Advisors that collaborate on search-order decisions. For example, one Advisor, might recommend choose the variable with maximum domain size while another recommends choose the variable with minimum domain size. The heart of ACE is FORR (FOr the Right Reasons), wich is a problem-solving and learning architecture for the development of expertise from multiple heuristics. ACE also learns each time it solves a problem best ways to confront it. FORR is equipped with a variety of weight-learning algorithms, permits the user to partition each task into stages, so that a weight-learning algorithm can learn weights for each stage in the solution process. Conceptually ACE works with advisors who possess a hierarchy of levels, at level one of the advisors recommend action, if not identify an action, control passes to level 2, which recommended a plan that contains several actions, if not identify an decision, control passes to level 3, in FORR all level-3 Advisors are heuristic and consulted in parallel. A decision is reached by combining their comments in a process called voting, if no decision is taken select one at random.

Autonomous Search (AS) [6], is a special case of adaptive systems whose objective is to improve its problem solving performance. A system AS changes the behavior of its internal components due to external forces and opportunities. Internal components are various algorithms involved in the search process, while external forces are information collected during the search process. Autonomous Solver can be understood as solvers that contain control in their resolution process either by:

- Self Adaptation: The adaptive mechanism is coupled with the search process, directly changing them in response to their actions.

- Supervised Adaptation: Work at a higher level. It is usually external, can use more information, for example learning-based knowledge.

Another work closely related is Automated Parameter Control for Evolutionary Algorithms [9] here outlined an approach to control automatically the parameters used in evolutionary algorithms. Evolutionary Algorithms are metaheuristics inspired by natural evolution that are used for solve optimisation problems. Broadly exists a population that is a set of possible solutions, exists two operators to create new solutions, mutation, that modifies randomly one part of the solution, and crossover, that combines the information of two of them. In addition exists a selection process that chooses the candidate solutions that will survive in the next generation population. The performances of these algorithms are strongly related to a suitable setting of several parameters such as population size and operators application rate. The tuning of these parameters is often done by a trial and error process therefore it requires too many repetitions, making expensive to find a configuration parameters, the complex interaction between parameters make difficult to understand all the relationship and where to find an optimal configuration, at each stage of the search requires different values for the parameters. The process described for the automatic adjustment of parameters consists of two phases. First one, Learning phase produces examples using different parameters values, to capture the mapping of these combinations, to generate another model in order to link parameters and quality, measured in terms of mean fitness. The second, Control phase, the controller modifies some parameters, in order to properly exploit space search and escape from local optima.

Adaptive Enumeration Strategies [10], proposes a framework, here adaptation consists in use information about solution process, during the search is collecting information about the state of progress, if no advancement adjustments must be made, changing the Enumeration Strategies. Information about the state of progress is captured through snapshots and indicators. Snapshots are observations about the current search tree while the indicators are the evidence of the resolution. Examples of snapshots are: the maximum
depth reached in the search tree, the depth of the current node, the size of the current search space. An example of an indicator is: variation of the maximum depth. The framework is formed with 4 components: the SOLVE component runs a generic CSP solving algorithm performing a depth-first search by alternating constraint propagation with enumeration phases. The OBSERVATION component aims at observing and recording some information about the current search tree, and take snapshots. The ANALYSIS component analyses the snapshots for evaluates the different strategies and provide indicators. The UPDATE component makes decisions using the indicators. It interprets the indicators, and then updates the enumeration strategies priorities and requests some metavbacktracks in the SOLVE component.

Sampling y Weighted Degree Heuristic [15] describes a heuristic known as Weighted Degree Heuristic, whose characteristic is to take sampling information during search to make more informed decisions. This approach also incorporates a method called random probing, this method suggests that instead of interlayer sampling and heuristics, sampling occurs at an initial stage where the variables are chosen at random and search is run repeatedly to a fixed cutoff. This solves the original problem in Weighted Degree Heuristic where the initial elections are often the most important and are made without information based on edge weights. The weighted degree heuristic is designed to enhance variable selection by incorporating knowledge gained during search. In this procedure, a constraints weight is incremented during arc consistency propagation whenever this causes a domain wipeout. This information is used during variable selection by calculating the sum of the weights of the constraints associated with a variable and choosing the variable with the largest sum. This constraint weight sum is referred to as a weighted degree and the heuristic for selecting a variable can therefore be called the weighted degree heuristic. The principles that support the weighted degree procedure, can be conceived in terms of an overall strategy that combines two heuristic principles, the fail-first and the contention principle. The fail-first principle says: to succeed, you must first search where you are most likely to fail [7] while contention principle says: that variables directly related to failure (domain wipeouts) are more likely to cause failure if they are chosen instead of other variables (sampling strategies are based on this principle).

In our work the adaptation consists in the use of information about the process during the search, collecting information about the state of progress in order to be able to react and change strategy as needed.

3 CP Resolution Technique

In the resolution of Constraint Satisfaction Problems diverse techniques can be used, currently they are solved using complete techniques (global optimization) and incomplete techniques (local optimization), and hybridizations of both techniques [5]. Specifically, the Constraint Programming community uses a complete approach alternating phases of Constraint Propagation and Enumeration, where the propagation prunes the search tree by eliminating values that can not participate in a solution. Enumeration consists of dividing the original CSP in two smaller CSPs, creating one branch by instantiating a variable \((x = v)\) and another branch \((x \neq v)\) for backtracking when the first branch does not contain any solution [1].

Combining the exploration of variables/values with a look-ahead strategy allows narrow the domains of the variables and reduces the remaining search space through constraint propagation. Here we use this technique named Forward Checking [11].

When enumerating two decisions have to be made: What variable is selected to be instantiated? and What value is assigned to the selected variable?. In order to support these decisions we use Ordering heuristics.

4 Enumeration Strategies

The enumeration strategies are constituted by the combination of Variable and Value Ordering Heuristics.

4.1 Variable Ordering Heuristics

Here we use the following variable ordering heuristics:

- First (F): the first variable of the list is selected.
- Minimum Remaining Values (MRV): at each step, the variable with the smallest domain size is selected.
- Anti Minimum Remaining Values (AMRV): at each step, the variable with the largest domain size is selected.
- Occurrence (O): the variable with the largest number of attached constraints is selected.

4.2 Value Ordering Heuristics

Here we use the following value ordering heuristics:

- In Domain (ID): it starts with the smallest element and upon backtracking tries successive elements until the entire domain has been explored.
Table 1: Search Process Indicators.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VFE</td>
<td>Number of Variables Fixed by Enumeration [10]</td>
</tr>
<tr>
<td>VFP</td>
<td>Number of Variables Fixed by Propagation [10]</td>
</tr>
<tr>
<td>VF</td>
<td>Number of Variables Fixed by Enumeration and Propagation [10]</td>
</tr>
<tr>
<td>SB</td>
<td>Number of Shallow Backtracks [2]</td>
</tr>
<tr>
<td>B</td>
<td>Number of Backtracks [4]</td>
</tr>
<tr>
<td>N</td>
<td>Number of Nodes visited by a procedure [4]</td>
</tr>
</tbody>
</table>

○ In Domain Max (IDM): it starts the enumeration from the largest value downwards.

5 Measuring The Search

Different indicators provide information that will enable us to know the state of progress in solution process a problem. The table 1 show a list of indicators used. The Shallow Backtrack [2]: During the search each time the algorithm attempts to assign a value to the variable, constraint propagation is performed and domains of non-instantiated variables are filtered. If any domain becomes empty then the assignment fails and a next value is tried. Backtrack: During the search when a dead-end situation is reached, move back to variable previously assigned to try assigning a new value.

6 Controlling CP Search with a Hyperheuristic

Variable and value ordering heuristics have various strengths and weaknesses, combine heuristics can compensate these differences. We organize the combination of heuristics based on the guidance of a hyperheuristic. Solving problems a hyperheuristic tries to automate the selection or combination of several simpler heuristics (or low level components). Then, it is a “heuristics to choose heuristics”. The main motivation for use hyper-heuristics is to build solvers which can handle wide classes of problems rather than solving just one problem. Hyper-heuristics search within a search space of heuristics (metaheuristics search within a search space of solutions). Thus, when using hyper-heuristics, we are attempting to find the right sequence of low level heuristics in a given situation rather than trying to solve a problem directly. Here the low level heuristic selection mechanism used is choice function.

The choice function attempts to capture the correspondence between decision point currently being investigated and the historical performance of each heuristic [13]. The choice function is used to rank and choose between different enumeration strategies at each step.

During the search, information about the state of progress is captured, for each enumeration strategy, information concerning the recent effectiveness of the enumeration strategy \( f_1 \), see formula 1, interpreted as revenue minus cost, and information concerning the amount of steps since enumeration strategy was last called \( f_2 \), see formula 3. The new value of \( f_1(N_j) \) can be calculated from the old value using the formula 2. The parameter \( \alpha \) is a value between 0 and 1, which reflects the importance attached to recent performance (Exploitation), and \( \beta \) is a parameter to control the importance of encourage the use the enumeration strategies that have not recently been used (Exploration). The choice function used in this work is showed in the formula 4.

\[
f_1(N_j) = \sum_n \alpha^{n-1}(VF_n(N_j) - SB_n(N_j)) \tag{1}
\]

\[
f_1(N_j) \leftarrow VF_1(N_j) - SB_1(N_j) + \alpha f_1(N_j) \tag{2}
\]

\[
f_2(N_j) = \tau(N_j) \tag{3}
\]

\[
f(N_j) = \alpha f_1(N_j) + \beta f_2(N_j) \tag{4}
\]
<table>
<thead>
<tr>
<th>Strategy</th>
<th>NQ n=8</th>
<th>NQ n=16</th>
<th>NQ n=20</th>
<th>MS n=3</th>
<th>MS n=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>F + ID</td>
<td>10</td>
<td>542</td>
<td>10026</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>F + IDM</td>
<td>11</td>
<td>542</td>
<td>10026</td>
<td>1</td>
<td>51</td>
</tr>
<tr>
<td>MRV + ID</td>
<td>10</td>
<td>3</td>
<td>11</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>MRV + IDM</td>
<td>10</td>
<td>3</td>
<td>11</td>
<td>1</td>
<td>97</td>
</tr>
<tr>
<td>AMRV + ID</td>
<td>11</td>
<td>517</td>
<td>2539</td>
<td>4</td>
<td>1191</td>
</tr>
<tr>
<td>AMRV + IDM</td>
<td>11</td>
<td>517</td>
<td>2539</td>
<td>0</td>
<td>42</td>
</tr>
<tr>
<td>O + ID</td>
<td>10</td>
<td>542</td>
<td>10026</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>O + IDM</td>
<td>10</td>
<td>542</td>
<td>10026</td>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>Dynamic</td>
<td>6</td>
<td>404</td>
<td>629</td>
<td>1</td>
<td>23</td>
</tr>
</tbody>
</table>

Table 2: Number of Backtracks solving N-Queens (NQ) and Magic Square (MS) with different strategies.

7 Experimental Results

Our implementation has been written with the ECLiPSe Constraint Programming System version 5.10. The cut-off time is ten minutes for each experiment. The table 2 presents the results of each enumeration strategy and the dynamic approach with $\alpha = 1$ and $\beta = 0.7$. Due space reasons we show only experimental results measured in number of backtracks but the conclusion is the same using other performance metrics (nodes visited, time): for each problem the dynamic approach gains very good position in the global ranking, it was the best for N-Queens n=8; it was the second for N-Queens n=16, N-Queens n=20 and Magic Square n=3; and it ranked fourth for Magic Square n=4.

8 Conclusion

We have presented a proposal for automated selection of enumeration strategies for constraint solving. Based on strategy performances, our dynamic approach is able to detect good cases using a choice function. We have applied a range of fixed enumeration strategies and our dynamic approach to solve different CSPs exhibiting good results. We plan to extend our work to self tuning of the choice function parameters.

References:


