Variable and Value Selection Heuristics: Application to Solve Puzzles

Broderick Crawford[1,2] and Mary Aranda[1]

[1] Pontificia Universidad Católica de Valparaíso, Chile
[2] Universidad Técnica Federico Santa María, Chile
broderick.crawford@ucv.cl
mary.aranda.c@mail.ucv.cl

Abstract. Constraint Programming is one of the major contributions of Computer Science for solving problems of high complexity. This problems can be modeled like Constraint Satisfaction Problems and solving them via Constraint Propagation and Enumeration. In this work, we study the resolution of puzzles (Latin Square, Magic Square and Sudoku)with this approach and we evaluate the performance of different variable and value selection heuristics in the Enumeration phase, demonstrating the relevance of the Enumeration Strategies in a Constraint Programming process.

1 Introduction

The Constraint Programming (CP) has been defined as a technology of Software used to describe and solve combinatorial problems [2, 3]. 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 [19]. The goal is to assign a value to each variable satisfying all the constraints. The most general notation for CSP is the following [21]: $< C; x_1 \in D_{x_1}, ..., x_n \in D_{x_n} >$

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.

The underlying structure to the described paradigm CP is shown in Figure 1.

© A. Gelbukh, Á. Kuri (Eds.)
Advances in Artificial Intelligence and Applications
Research in Computer Science 32, 2007, pp. 238–248

Received 17/06/07
Accepted 31/08/07
Final version 27/09/07

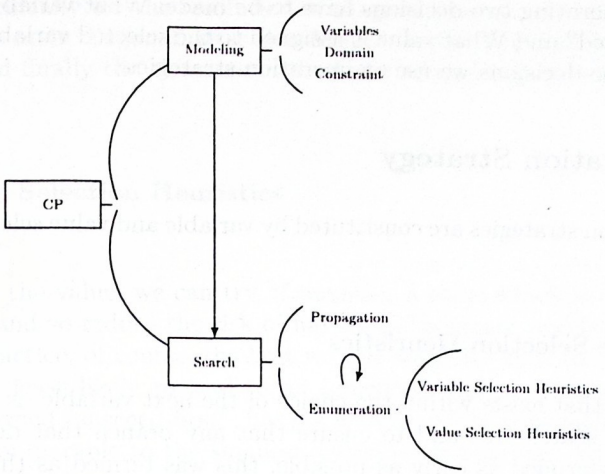

Fig. 1. Constraint Programming Structure

This work is focused on the Enumeration phase of CP, where the use of variable and value selection heuristics is critical. A suitable definition and the use of an enumeration strategy can improve the resolution process strongly. We apply to resolution of puzzles (Magic Square, Latin Square and Sudoku) different variables and values selection heuristics presented in the literature [13, 17, 4, 6].

This paper is organized as follows: in Section 2 is described the resolution technique, Section 3 and 4 show the enumeration strategies and the modeling of each problem in the platform used: Mozart[1]. In Section 5, we analyze the results obtained and finally we conclude in Section 6.

2 Resolution Technique

In the resolution of Constraint Satisfaction Problems diverse techniques can be used, currently they are solved using complete techniques (global optimization), incomplete techniques (local optimization), and hibridizations of both techniques [8]. Specifically, the Constraint Programming community uses a complete approach alternating phases of constraint propagation and enumeration [7], where the propagation prunes the search tree by eliminating values that can not participate in a solution. Enumeration [1] consists of dividing the original CSP in two smaller CSPs, creating one branch by instantiating a varible $(x = v)$ and another branch $(x \neq v)$ for backtracking when the first branch does not contain any solution.

[1] www.mozart-oz.org

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 enumeration strategies.

3 Enumeration Strategy

The enumeration strategies are constituted by variable and value selection heuristics [12].

3.1 Variable Selection Heuristics

The main idea that exists within the choice of the next variable, is to minimize the size of the search tree and to ensure that any branch that does not lead to a solution is pruned as early as possible, this was termed as the "fail-first" principle by Haralick and Elliot [9], described as *"To succeed, try first where you are most likely to fail"* [16, 1].

For variable selection, it is possible to find a classification according to the moment when the selection order is done.

Static Selection Heuristic: it generates a fixed order of the variables before initiating the search. Here the variables are always selected in the order predefined for instantiation.

Dynamic Selection Heuristic: it can change the instantiation order of the variables dynamically as one advances in the tree search. It is based on information generated during the search.

Here the dynamic and static terms are used according to the definition given in [2], also corresponding to the used in [16]. This definition differs from the concept used in [7, 12], where the idea of dynamism is based on adaptive constraint satisfaction [5], where the idea of dynamic selection heuristic consists basically in detecting the bad decisions made with respect to the selection of variables and values, replacing on-the-fly the strategies with bad results by others that promise to be better.

In this work we used the following variable selection heuristics:

Minimum Domain Size (*MiD*): at each enumeration step the domain of each one of the variables not yet instantiated is analyzed, then the variable with smaller domain size is selected.
Maximum Domain Size (*MaD*): the idea of this heuristic is similar to the previous one, nevertheless in this case it selects the variable with the greater domain size.
Order Previously Established (*Opre*): in this static heuristic, we fix a variable order before initiating the resolution process. The heuristic *Opre* has been

defined specifically for the resolution of magical squares, selecting first the elements of the main diagonal, after the elements in the upper triangle of the main diagonal and finally the elements in the lower triangle of the main diagonal.

3.2 Value Selection Heuristics

In choosing the value, we can try, if possible, a value which is likely to lead to a solution, and so reduce the risk of having to backtrack and try an alternative value. In practice, of course, the best we can normally do is to choose the value which seem least likely to lead to an inmediate failure. This principle, which might be termed "succeed-first", has not lead to widely-applicable value ordering heuristics comparable to the smallest-domain-first heuristic, but can give good heuristics tailored to individual problems, or types of problem [16]. In synthesis, this principle (usually problem dependent) indicates that the value with a high number of supports is likely to be preferred.

In this work we used the following value selection heuristics:

Smaller Value of the Domain (*SVal*): this heuristic establishes that the smallest value of the domain is always chosen.

Greater Value of the Domain (*GVal*): it is similar to the previous one, but instead of choosing the smallest element of the domain, the greater element is selected.

Average Value of the Domain (*AVal*): this heuristic selects the value of the domain that is more near to the half of the domain, it calculates the arithmetic average between the limits (superior and inferior) of the domain of the selected variable and in case of having a tie the smallest value is selected. In order to understand better this concept: considering that x_1 has been selected to the variable whose domain is 2#4, to select the value to instantiate calculates the average between the limits of the domain, this is:$(2 + 4)/2 = 3$, therefore the value that is selected in this stage corresponds to 3. If the domain of x_1 went 1#4 the value to select he would be 2, since the average is $(1 + 4)/2 = 2.5$ and in case of tie always the smaller value is selected.

Immediately Greater Value to the Average Value of the Domain (*GAV*): this heuristics selects the smaller value of the domain that it is greater as well to the average value of the domain. Consider the previous example, when the domain of the variable is 2#4 in heuristic the previous one selects value 3, however with present the heuristic one selects value 4. Thus, when the domain of the variable is 1#4 present the heuristic selects value 3.

Finally, established the heuristic to use, the enumeration strategies are compound according to Table 1.

S_1 = MiD+SVal	S_5 = MaD+SVal	S_9 = Opre+SVal
S_2 = MiD+GVal	S_6 = MaD+GVal	S_{10} = Opre+GVal
S_3 = MiD+AVal	S_7 = MaD+AVal	S_{11} = Opre+AVal
S_4 = MiD+GAV	S_8 = MaD+GAV	S_{12} = Opre+GAV

Table 1. Enumeration Strategies

4 Description of Problems

4.1 Magic Square

This puzzle consists in finding for a given N an NxN matrix such that every cell of the matrix is a number between 1 and N^2, all the cells of the matrix must to be different, and the sum of the rows, columns, and the two diagonals are all equal.

The mathematical model used for its representation defines a variable x_{ij} that represents the value that each cell of the matrix can take, and a variable S for the sum of each row, column and diagonal. Then the CP model establishes the following constraint:

$$\forall i,j \in \{1,...,N\} \;\; Alldifferent\{x_{ij}\} \tag{1}$$

$$\sum_{j=1}^{N} x_{ij} = S \;\; \forall i \in \{1,...,N\} \tag{2}$$

$$\sum_{i=1}^{N} x_{ij} = S \;\; \forall j \in \{1,...,N\} \tag{3}$$

$$\sum_{i=1}^{N} x_{ii} = S \tag{4}$$

$$\sum_{i=1}^{N} x_{i(N-i+1)} = S \tag{5}$$

The constraints (2) and (3) ensure that the sum of each row and each column will be equal to S, and the constraints (4) and (5) assure that the sum of each diagonal will be equal to S.

4.2 Latin Square

A Latin Square puzzle of order N is defined as an NxN matrix where all its elements are numbers between 1 and N with the property that each one of the N numbers appear exactly once in each row and exactly once in each column of the matrix.

The mathematical representation used to model the problem has a variable x_{ij} that represents the value of the cell (i, j) of the matrix. The CP model consists of the following constraints:

$$\forall i \in \{1, ..., N\} \quad Alldifferent\{x_{i1}, x_{i2}, ..., x_{iN}\} \tag{6}$$

$$\forall j \in \{1, ..., N\} \quad Alldifferent\{x_{1j}, x_{2j}, ..., x_{Nj}\} \tag{7}$$

4.3 Sudoku

Sudoku is a puzzle played in a 9x9 matrix (standard sudoku) which, at the beginning, is partially full. This matrix is composed of 3x3 submatrices denominated "regions". The task is to complete the empty cells so that each column, row and region contain numbers from 1 to 9 exactly once [10, 11].

The model used for the representation can be seen like a composition of the models used in the above puzzles, the variable x_{ij} represents the value that each cell (i, j) can take (in this case, from 1 to 9). In order to restrict that each row and each column have values from 1 to 9 exactly once the following constraint are due to impose:

$$\forall i \in \{1, ..., 9\} \quad Alldifferent\{x_{i1}, x_{i2}, ..., x_{i9}\} \tag{8}$$

$$\forall j \in \{1, ..., 9\} \quad Alldifferent\{x_{1j}, x_{2j}, ..., x_{9j}\} \tag{9}$$

On the other hand, each cell in regions S_{kl} with $0 \leq k, l \leq 2$ must be different, which forces to include in the model the following constraint:

$$\forall i, j \quad Alldifferent\{x_{ij}, x_{i(j+1)}, x_{i(j+2)}, x_{(i+1)j}, x_{(i+1)(j+1)}, x_{(i+1)(j+2)}, \tag{10}$$

$$x_{(i+2)j}, x_{(i+2)(j+1)}, x_{(i+2)(j+2)}\}$$

con $i = k * 3 + 1$ y $j = l * 3 + 1$.

5 Analysis of Results

Each one of the exposed problems were solved using the eight first strategies listed in Table 1, and additionally the strategies $S_9, ..., S_{12}$ were used in the resolution of magic squares puzzles, because these strategies are constituted by a variable selection heuristic designed specifically for such problem.

The tests conducted allow to evaluate the performance of the enumeration strategies based on the following indicators of performance:

Number of Backtracking (B): it shows the amount of bad decisions made during the search of the solution, that is calculations or decisions executed without leading to a solution.

Number of Enumerations (E): this metric tells the amount of nodes or spaces generated to find the solution of the problem, including the good enumerations that lead to a solution and the bad enumerations that force to backtrack.

Time (t): it measures the required time to solve the problem.

Each execution had a time limited to 10 minutes, not finding results are indicated with the symbol "-".

5.1 Searching the First Solution

In general it is possible to appreciate that for small instances the strategies do not reflect significant differences. Nevertheless when observing the results obtained for each one of the problems it is possible to see that the performance of the strategies varies as N increases, this because with the increase of N the size of the search space grows drastically [20]. On the other hand, when observing the results obtained it is perceived that the strategies constituted by the heuristic *MiD* $(S_1, ..., S_4)$ have better behavior in those instances in which the search space grows, this in comparison with strategies that are guided by the heuristic *MaD* $(S_5, ..., S_8)$. Such differences happen mainly because the heuristic *MiD* leads as rapidly as possible to an insolvent space, allowing to prune the tree search. Leading to an insolvent space quickly consists of choosing variables with few elements in its domain, increasing the probability of failing before generating a big search tree.

N	3			4			5			10			15		
	(E)	(B)	(t)	(E)	(B)	(t)	(E)	(B)	(t)	(E)	(B)	(t)	(E)	(B)	(t)
S_1	3	0	9	6	0	10	10	0	10	67	1	18	165	7	76
S_2	3	0	10	6	0	10	10	0	10	67	1	18	165	7	78
S_3	3	0	10	6	0	10	12	0	12	70	2	18	163	5	67
S_4	3	0	10	6	0	10	10	0	11	70	4	31	309	138	229
S_5	3	0	12	8	0	10	94	77	16	-	-	-	-	-	-
S_6	3	0	9	8	0	10	94	77	15	-	-	-	-	-	-
S_7	3	0	10	8	0	10	1644	1625	106	-	-	-	-	-	-
S_8	3	0	10	8	0	10	36	21	12	-	-	-	-	-	-

Table 2. Latin Squares: Enumerations (E), Backtracking (B), CPU time (t) in ms.

Observing the results obtained for the magic square problem, one of the aspects to emphasize is the good performance obtained by the strategies constituted by the heuristic *Opre*, particularly using it in combination with value selection heuristic *AVal* and *GAV*, this good behavior reflected in the Table 3, it is due mainly because the heuristic *Opre* generates a more effective constraint propagation, reducing the size of the search space considerably. The reduction

N^2	9			16			25			49		
	(E)	(B)	(t)	(E)	(B)	(t)	(E)	(B)	(t)	(E)	(B)	(t)
S_1	4	1	1	741	717	30	208861	208827	12669	-	-	-
S_2	13	7	1	1681	1655	63	9113251	9113227	505231	-	-	-
S_3	8	5	1	2465	2449	81	2187	2158	76	101510	101424	4415
S_4	13	7	1	406	384	27	7061	7037	452	7139	7081	487
S_5	16	7	1	45097	45043	3680	-	-	-	-	-	-
S_6	14	8	1	681	646	46	-	-	-	-	-	-
S_7	22	10	1	847094	847055	48519	-	-	-	-	-	-
S_8	23	19	1	358775	358772	27115	557090	557608	44189	-	-	-
S_9	5	2	0	49	34	3	26475	26428	1396	-	-	-
S_{10}	13	7	1	508	491	24	80761	80728	5078	-	-	-
S_{11}	8	5	1	1323	1296	56	393	375	21	-	-	-
S_{12}	13	7	1	791	763	55	12	1	2	82107	82047	6649

Table 3. Magic Squares: Enumerations (E), Backtracking (B), CPU time (t) in ms

		S_1			S_2			S_3			S_4		
Source	Degree	(E)	(B)	(t)	(E)	(B)	(t)	(E)	(B)	(t)	(E)	(B)	(t)
[14]	None-1	84	52	14	220	195	21	1308	1283	88	183	159	26
[14]	None-2	2836	2815	153	271	249	23	11074	11048	603	124	102	22
[18]	Easy	7	3	11	17	13	11	7	3	10	17	13	12
[18]	Medium	16	6	11	174	164	19	16	6	11	174	164	26
[18]	Hard	27	16	11	24	18	11	27	16	11	24	18	12

Table 4. Sudoku solved with heuristic *MiD*

		S_5			S_6		
Source	Degree	(E)	(B)	(t)	(E)	(B)	(t)
[14]	None-1	-	-	-	-	-	-
[14]	None-2	-	-	-	-	-	-
[18]	Easy	18554	18537	1799	274476	274472	28149
[18]	Medium	-	-	-	121135	121113	12868
[18]	Hard	-	-	-	-	-	-

Table 5. Sudoku solved with S_5 and S_6 strategy

of the search space takes place because at the beginning of the process the variables chosen are related to a greater number of other variables (elements of the main diagonal), which produces that when evaluating the levels of consistency between the variables is eliminated a greater amount of inconsistency, reducing the space to explore.

In order to conclude, it is possible to mention that the size of the search space has a great incidence in the resolution process, where his exponential growth makes the process considerably more expensive, this is possible to appreciate when solving different instances of the same problem (of course, hypothesis demonstrated in a lot of previous work)

		S_7			S_8		
Source	Degree	(E)	(B)	(t)	(E)	(B)	(t)
[14]	None-1	-	-	-	-	-	-
[14]	None-2	-	-	-	-	-	-
[18]	Easy	24195	24169	2582	721773	72155	7484
[18]	Medium	-	-	-	88720	88706	9763
[18]	Hard	93138	93105	9158	-	-	-

Table 6. Sudoku solved with S_7 and S_8 strategy

With regard to the resolution of Sudoku, different published instances have been used from puzzles in the newspaper "The Times" [18], where usually the difficulty of puzzle is provided. On the other hand also a collection [14] of puzzles have been used which have single 17 numbers given in the initial matrix, this amount corresponds to the smaller known number given in the initial matrix [15], these last ones are not classified by difficulty.

In Table 4, Table 5 and Table 6 we show the source from where puzzles were obtained, the degree (difficulty level), the heuristic used to solve each instance and the different measured indicators of performance during the execution of the tests. Although some authors say that the amount of numbers given initially does not have incidence in the degree of difficulty of Sudoku, the results obtained here show bad results with 17 numbers in comparison with the other cases (easy, medium and hard), where the amount of numbers given initially is greater 25.

6 Conclusions

In this work we showed that variable and value selection heuristic influence the efficiency in the resolution of combinatorial problems. The efficiency of resolution was measured on the basis of performance indicators. The work included the modeling and resolution of classic puzzles (Magic Square, Latin Square and Sudoku) in Mozart.

The possibility to obtain better results in the search process was showed using suitable criteria of selection of variables and values. In fact, to select a variable in

a search process implies to determine the descending nodes of the present space that have a solution. It is very important to detect when the descending nodes are not a solution, because in this way we avoided to do unnecessary calculations that force to backtracking. Due to the above reasoning the heuristic selecting the variable with minimum domain size (*MiD*) presents a better behavior in comparison with the other strategies, because *MiD* bets by the variable going towards an insolvent space avoiding a priori unnecessary calculations.

We showed that the resolution possibilities of a certain problem depends on the search space size, it is possible to be appreciated when observing the differences generated in the results obtained in the resolution of different instances of the same problem.

At the moment of thinking in selection heuristics is important to consider the "model-and-run" paradigm for CP, in order to use variable and value selection heuristic efficiently.

In the future, we plan to integrate in a Constraint Programming process other techniques such as Local Search or Metaheuristics working like enumeration strategies.

References

1. K. Apt. Principles of constraint programming, 2003.
2. F. Barber and M. A. Salido. Introducción a la programación de restricciones. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 20:13–30, 2003.
3. R. Bartk. On-line guide to constraint programming. 1998. http://kti.mff.cuni.cz/ bartak/constraints/.
4. C. Beck, P. Prosser, and R. Wallace. Toward understanding variable ordering heuristics for constraint satisfaction problems. In *Fourteenth Irish Artificial Intelligence and Cognitive Science Conference - AICS 2003*, pages 11–16.
5. J. Borrett, E. Tsang, and N. Walsh. Adaptive constraint satisfaction: The quickest first principle. In W. Wahlster, editor, *ECAI*, pages 160–164. John Wiley and Sons, Chichester, 1996.
6. M. Cadoli, T. Mancini, and F. Patrizi. Sat as an effective solving technology for constraint problems. In F. Esposito, Z. Ras, D. Malerba, and G. Semeraro, editors, *ISMIS*, volume 4203 of *Lecture Notes in Computer Science*, pages 540–549. Springer, 2006.
7. C. Castro, E. Monfroy, C. Figueroa, and R. Meneses. An approach for dynamic split strategies in constraint solving. In A. F. Gelbukh, A. de Albornoz, and H. Terashima-Marín, editors, *MICAI*, volume 3789 of *Lecture Notes in Computer Science*, pages 162–174. Springer, 2005.
8. C. Castro, M. Moossen, and M.-C. Riff. A cooperative framework based on local search and constraint programming for solving discrete global optimisation. In A. Bazzan and S. Labidi, editors, *SBIA*, volume 3171 of *Lecture Notes in Computer Science*, pages 93–102. Springer, 2004.
9. R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263 - 313, 1980.
10. T. Lambert, E. Monfroy, and F. Saubion. Solving sudoku with local search : A generic framework, May 28-31 2006. A paratre.

11. R. Lewis. Metaheuristics can solve sudoku puzzles. *In Press: Journal of heuristics*, 13, 2007.
12. E. Monfroy, C. Castro, and B. Crawford. Adaptive enumeration strategies and metabacktracks for constraint solving. In T. M. Yakhno and E. J. Neuhold, editors, *ADVIS*, volume 4243 of *Lecture Notes in Computer Science*, pages 354–363. Springer, 2006.
13. E. Monfroy, C. Castro, and B. Crawford. Using local search for guiding enumeration in constraint solving. In J. Euzenat and J. Domingue, editors, *AIMSA*, volume 4183 of *Lecture Notes in Computer Science*, pages 56–65. Springer, 2006.
14. G. Royle. Minimum sudoku. http://www.csse.uwa.edu.au/ gordon/sudoku17.
15. H. Simonis. Sudoku as a constraint problem. In B. Hnich, P. Prosser, and B. Smith, editors, *Proc. 4th Int. Works. Modelling and Reformulating Constraint Satisfaction Problems*, pages 13–27, 2005.
16. B. Smith. Succeed-first or Fail-first: A Case Study in Variable and Value Ordering. Technical Report 96.26, 1996.
17. B. M. Smith and P. Sturdy. Value ordering for finding all solutions. In L. P. Kaelbling and A. Saffiotti, editors, *IJCAI*, pages 311–316. Professional Book Center, 2005.
18. The Times. Spring su doku. April 2007.
19. E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London, 1993.
20. T. Yato and T. Seta. Complexity and completeness of finding another solution and its application to puzzles. *IPSJ SIG Notes*, 2002(103):9–16, 20021108.
21. P. Zoeteweij. *Composing Constraint Solvers.* Printed and bound by PrintPartners Ipskamp, Enschede, 2005.