Solving Sudoku with Constraint Programming

Chapter *in* Communications in Computer and Information Science · January 2009

DOI: 10.1007/978-3-642-02298-2_52

CITATIONS
6

READS
4,650

3 authors:

Broderick Crawford
Pontificia Universidad Católica de Valparaíso
367 PUBLICATIONS **3,056** CITATIONS
SEE PROFILE

Carlos Castro
Universidad Técnica Federico Santa María
138 PUBLICATIONS **1,278** CITATIONS
SEE PROFILE

Eric Monfroy
University of Nantes
258 PUBLICATIONS **1,948** CITATIONS
SEE PROFILE

Solving Sudoku with Constraint Programming

Broderick Crawford[1], Carlos Castro[2], and Eric Monfroy[3]

[1] Pontificia Universidad Católica de Valparaíso, Chile
and
Universidad Técnica Federico Santa María, Chile
FirstName.Name@ucv.cl
[2] Universidad Técnica Federico Santa María, Chile
FirstName.Name@inf.utfsm.cl
[3] LINA, Université de Nantes, Nantes, France
and
Universidad Técnica Federico Santa María, Valparaíso, Chile
FirstName.Name@univ-nantes.fr

Abstract. Constraint Programming (CP) is a powerful paradigm for modeling and solving Complex Combinatorial Problems (generally issued from Decision Making). In this work, we model the known Sudoku puzzle as a Constraint Satisfaction Problems and solve it with CP comparing the performance of different Variable and Value Selection Heuristics in its Enumeration phase. We encourage this kind of benchmark problem because it may suggest new techniques in constraint modeling and solving of complex systems, or aid the understanding of its main advantages and limits.

1 Introduction

The Constraint Programming has been defined as a technology of Software used in complex system modeling and combinatorial optimization 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. Constraint Programming community uses a complete search 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 [Apt, 2003]. 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, then the enumeration strategies are constituted by variable and value selection heuristics [Monfroy et al., 2006].

2 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"

Y. Shi et al. (Eds.): MCDM 2009, CCIS 35, pp. 345–348, 2009.

Table 1. Enumeration Strategies = Variable + Value Selection Heuristics

S_1 = MiD+SVal	S_2 = MiD+GVal	S_3 = MiD+AVal	S_4 = MiD+GAV
S_5 = MaD+SVal	S_6 = MaD+GVal	S_7 = MaD+AVal	S_8 = MaD+GAV

principle by Haralick and Elliot [Haralick and Elliot, 1980], described as *"To succeed, try first where you are most likely to fail"* [Smith, 1996]. In this work we used the following 2 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; and **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.

3 Value Selection Heuristics

In choosing the value, we can try, if it is 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 ("succeed-first" principle[Smith, 1996]). In this work we used the following 4 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; and **Immediately Greater Value to the Average Value of the Domain (*GAV*):** this heuristic selects the smaller value of the domain that it is greater as well to the average value of the domain. Finally, established the heuristics to use, the enumeration strategies are compound according to Table 1.

4 Constraint-Based Model of 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 [Simonis, 2005]. The CP model consists of the following constraints:

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

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

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:

$$\forall i,j \quad Alldifferent\{x_{ij}, x_{i(j+1)}, x_{i(j+2)}, x_{(i+1)j}, \tag{3}$$
$$x_{(i+1)(j+1)}, x_{(i+1)(j+2)}, x_{(i+2)j}, x_{(i+2)(j+1)}, x_{(i+2)(j+2)}\}$$
$$\text{con } i = k*3+1 \quad \text{y } j = l*3+1.$$

Table 2. Sudoku solved with heuristic *MiD*

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

Table 3. Sudoku solved with heuristic *MaD*

		S_5			S_6			S_7			S_8		
S.	D.	(E)	(B)	(t)	(E)	(B)	(t)	(E)	(B)	(t)	(E)	(B)	(t)
		-	-	-	-	-	-	-	-	-	-	-	-
		-	-	-	-	-	-	-	-	-	-	-	-
		18554	18537	1799	274476	274472	28149	24195	24169	2582	721773	72155	7484
		-	-	-	121135	121113	12868	-	-	-	88720	88706	9763
		-	-	-	-	-	-	93138	93105	9158	-	-	-

5 Analysis of Results

The benchmark problems were implemented and solved in the platform Mozart[1] with the strategies listed in Table 1. Results are showed in Tables 2 and 3, each execution had a time limited to 10 minutes, not finding results are indicated with the symbol "-". The performance evaluation was based on the following known indicators in constraint solving: Number of Backtracks (B), Number of Enumerations (E), or Nodes Visited, and Time (t). 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. Different published instances have been used from The Times[2] and Minimum Sudoku page[3].

[1] www.mozart-oz.org
[2] http://entertainment.timesonline.co.uk
[3] http://people.csse.uwa.edu.au/gordon/sudokumin.php

6 Conclusions

In this work we showed that variable and value selection heuristics influence the efficiency in the resolution of Sudoku in Mozart. The efficiency of resolution was measured on the basis of performance indicators. 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 early when the descending nodes are not in a solution, because in this way we avoided to do unnecessary calculations that force to backtracking.

Acknowledgements. The second author has been partially supported by the Chilean National Science Fund through the project FONDECYT 1070268. The third author has been partially supported by Escuela de Ingeniería Informática PUCV through the project INF-03/2008 and DGIP-UTFSM through a PIIC project.

References

Apt, K.: Principles of constraint programming (2003), http://citeseer.ist.psu.edu/apt03principles.html

Haralick, R., Elliot, G.: Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence 14, 263–313 (1980)

Monfroy, E., Castro, C., Crawford, B.: Adaptive enumeration strategies and metabacktracks for constraint solving. In: Yakhno, T., Neuhold, E.J. (eds.) ADVIS 2006. LNCS, vol. 4243, pp. 354–363. Springer, Heidelberg (2006)

Simonis, H.: Sudoku as a constraint problem. In: Hnich, B., Prosser, P., Smith, B. (eds.) Proc. 4th Int. Works. Modelling and Reformulating Constraint Satisfaction Problems, pp. 13–27 (2005), http://4c.ucc.ie/~brahim/mod-proc.pdf

Smith, B.: Succeed-first or Fail-first: A Case Study in Variable and Value Ordering. Technical Report 96.26 (1996), http://citeseer.ist.psu.edu/194952.html