Introduction

In this thesis we will compare several constraint solvers using the point of view of user who is not experienced in constraint programming. We will focus on easiness of learning process of each solver and measure performance using benchmarks which tests various aspects of examined systems.

# Motivation

Constraint programming is a programming paradigm which uses constraints to a describe solution rather than to program a way of achieving such solution. Constraint can be any condition which can be asserted as true or false – X < Y, Billy is older than Johnny, Z = 5 etc. As an example of problem which can be solved using constraint programming we will use Sudoku puzzle. Sudoku is worldwide known logical problem which is easy to explain, its difficulty can be scaled and one does not need previous training to solve Sudoku. It makes this problem easy to understand for many people and therefore it is very popular. Simple rules of Sudoku are: There is given table of size nine times nine. Every field of table contains number in range one to nine. In each column all numbers are different (this enforces that every column contains all numbers in range one to nine). In each row there are also all numbers different. Finally the same rule which applies for columns and rows stands for three times three sized squares which are in the puzzle marked with bolder lines. The Sudoku is prefilled with a couple of values. These values help at the beginning of solving and by their count and placement can be adjusted the difficulty.

// vlozit obrazek sudoku

The way how to describe Sudoku puzzle in the constraint programming is very straightforward. We will define following constraints:

1. There are 81 variables *s* which can contain values in range 1 to 9. We will arrange them in a two-dimensional array with size 9 × 9. (The Sudoku is table sized 9×9 containing values in a range 1..9)
2. For all *i* in 1..9 is true: All values of *si®* are different (Values in each row are different)
3. For all *j* in 1..9 is true: All values of *s®j* are different (Values in each column are different)
4. For each square is true: For all *k,l* such that *k,l* is in square values of skl are different (Values in each square are different)
5. For all prefilled values: qmn = V if and only if field with placed in column n and row m is prefilled and contains V.

These constraints fully describe Sudoku puzzle problem and as the reader can see does not differ from the commonly known rules. Person solving Sudoku puzzle can use many techniques starting with randomly filling the table and looking if this is good solution (algorithm using this technique is called GAT – Generate and Test) and ending with generating all possible fillings and repairing the solution if something fails (this algorithm is called backtracking). First approach can miss correct solution. Since second approach systematically searches the possible solutions it has to result in correct solution. However it can last enormous time to complete it (even on supercomputer). The secret of successful solution is in fact that not all numbers can be filled in specific field. If there is prefilled value 8 at position [6,7] it means that in row 6 and in column 7 cannot be another number 8. And because of constraint (4) there cannot be 8 also in right middle square. Person who does these observations usually writes to the destination field all possible values and as examination of the puzzle continues there are less and less possibilities to fill in. In easy Sudoku there is after this examination at least one field which can be filled with only one number. After filling all such fields solving continues in the same way until whole table is filled. Program using constraint programming solves it in the same way. For each variable it remembers the range of possible values (we will call it domain). Before the program start searching of solution it tries to eliminate as much values from domain as possible. It can reveal that problem does not have solution (if there is some variable with empty domain) before backtracking. There is no surprise that in user guide to Choco system is stated “if you know Sudoku, then you know constraint programming.”

# Constraint programming

Constraint programming of course consists of more techniques but generally it works just like person solving Sudoku described in previous paragraph. Since time complexity of searching in possible solutions can be enormous the most important is to eliminate as many solution candidates as possible before actual search. Unfortunately eliminating values from domain based on given constraints does not guarantee that model can be directly solved. This means we have to search the problem space for possible solution. Standard backtracking does not work efficiently. It does not have memory so it cannot use results which were computed in different branch. It case of fail it also cannot identify problematic evaluation of constraint so it has to search whole subtree of given branch to find that it cannot result in solution. Constraint programming offers techniques backmarking and backjumping which can identify the dead end of current branch and recheck evaluating of that variable instead of useless searching through whole subtree. Algorithms like looking ahead and forward checking can identify future problems so the search algorithm can eliminate bad decisions as soon as possible. Detailed description of these techniques would fill whole book and this thesis has another purpose. The reader who wants to learn more can found more in (citace Online guide to CSP).

From previous paragraphs reader could think that constraint programming is used only as an academic toy for solving Sudoku and other applications which are useless for real life. In fact constraint programming is used in various applications. A few examples of many contains scheduling, image recognition, financial modeling, planning, vehicle routing, configuration, computer networks and bioinformatics. Constraint programming was also successfully used at NASA in Deep Space 1 experiment. Deep Space 1 was a space probe using 12 cutting-edge technologies which were never tested in space before. One of these technologies was remote agent used to plan actions of space vehicle while only general commands were sent to agent. Agent used for planning constraint solver [citace <http://nmp.nasa.gov/ds1/tech/autora.html>].

# Constraint solvers

Programmer who wants to solve problem using constraint programming can use in his algorithms ideas described in previous paragraphs or use specialized software – a constraint solver. Constraint solver is system which uses constraint programming techniques to solve given problem. There are many solvers available both commercial and freeware. As a short list of available systems can be found in table 1.1:

|  |  |  |  |
| --- | --- | --- | --- |
| **Solver** | **Used programming language** | **Operating system** | **Note** |
| ECLIPSE | Prolog | \*nix, Windows | OSS |
| Minion | Own input language | \*nix, Windows | OSS |
| SICStus Prolog | Prolog | \*nix, Windows | Commercial, trial available |
| Choco | Java | OS independent, library | OSS |
| OPL | Own language |  | Commercial, trial available |
| Mozart | Oz | \*nix, Windows | OSS |
| Microsoft Excel | See note | Mac, Windows | Application dialog which tries find appropriate values of fields to satisfy wanted result variable. In Excel 2000 available in *Tools > Solver* menu |
| Gecode | C++ | OS independent, library | OSS |
| Disolver | C++ | ? | Commercial?, need to contact author |
| --- provest vyzkum --- |  |  |  |

Table 1.1

# Related work

There exists a paper “A Comparative Study of Eight Constraint Programming Languages Over the Boolean and Finite Domains” by A. Fernandez and others (citace fernandez00) which covers similar area. However this paper does not focus on the user experience with the solvers. They compared solvers by their performance on various benchmarks and discussed implementation of self referential quiz in each solver. We are not benchmarking the same sets of solvers and we do not use similar methodology which means this thesis conclusions cannot replace or update results from [fernandez00].

# Outline of the thesis

We described our motivation for this thesis and listed some constraint solvers. In the second chapter we will define methodology used to examine some of mentioned solvers. The examination consists of two parts – performance tests and usability tests. In third chapter we will define benchmarks used to performance tests. Fourth chapter describes in details each solver, mentions a little from their history but mainly focus on usability and easiness of learning and using the solver. In fifth chapter we will discuss the performance tests results and compare the solvers. Finally in sixth chapter we will state conclusion of whole examination process and present decision graph “which solver use in which situation”.