Methodology

In the introduction we explained what constraint solvers are and presented several examples of them. In the rest of the thesis we will focus on seven of them – ILOG OPL, SICStus Prolog, Mozart, ECLIPSE, Gecode, Choco and Minion. First two are professional commercial solutions the others are freely available open source products. Purpose of this thesis is to help new user with choosing the right solver. Therefore we will study hardness of learning and using of each solver and their performance and abilities. We tested solvers which use various programming languages and paradigms. Imperative paradigm is represented by C++ library Gecode and JAVA library Choco. For users experienced in logical programming might be interesting SICStus Prolog or ECLIPSE. Mozart is implementation of Oz which is multi-paradigm programming language. For these solvers user could use his existing experience and only learn API of the constraint library. The remaining solvers are configured by solver specific problem description language. This fact is advantage and disadvantage at once. As a disadvantage we must accept that user cannot use his experience with existing programming languages and has to learn new concepts. However specialized language for describing constraint problems can be more accessible for users which do not have programming experiences but needs to solve given problem.

We will examine all solvers using point of view of user experienced in given programming language but inexperienced in using of the solver. In case of OPL and Minion we will expect that user has general computers knowledge and is able to describe given problem in constraints. The first examination will try to answer the question how difficult it is to learn using the solver. We will model problems described in the third chapter and look for constraints which cannot be modelled and describe possible solutions. Important criterion is the quality of documentation. Solver can be the best of all but if the user cannot understand the usage it is useless. The quality of documentation is perceived subjectively and cannot be measured exactly. This means that any evaluation is only informational but should be considered. We will count as documentation the user guide, all other available guides, documents, web pages, doxygen style documentation. Availability of user forums or mailing list is also important part of learning of new system. We will mention some of them if there exists some for the solver. Last but not least topic is debugging. There are two areas which can be debugged - the correctness of program and correctness of model. Correctness of program means that program do what should do, that it handles all inputs as programmer expects and so on. Correctness of model means that model is describing given problem accurately. User should be able to inspect variables, visualise decision tree of search and lots of other information. We will discuss ways how solver informs about mistakes (and how descriptive the information is), tools provided with solver to debug the program and similarly the tools which can be used to debug model correctness.

When user masters the solver and uses it to solve real problems the time and space efficiency of used algorithms matters. We will not examine source codes and analyze time complexity of used algorithms. Instead we will measure the time needed to load and time to solve the problem. If the solver cannot provide such information we will measure only total time. We will also measure amount of consumed memory during the program execution. All measurements will be performed several times and averaged to avoid randomness. The robustness test will be also performed. In robustness test we will set limit one hour and try to compute the biggest magic sequence in given time. We will use models created in process described in previous paragraph. In \cite{fernandez00} authors sent models to the solvers authors and gave them chance to modify them to achieve the best performance of their solvers. We focus on first time users of the solver so we will use our own models which are not perfect and tuned especially for particular solver. All solvers but OPL will be tested on Debian 4.0 Linux with kernel 2.6.18 on Pentium 4, 3GHz (single core with hyper threading). Due to licence problems OPL will be tested on Windows XP SP2 on Intel Core Duo, 1.6GHz. For comparison we will test on this platform also ECLIPSE which should give us relative comparison with solvers tested on another computer. Vzhledem k omezenim na OPL v testovaci verzi, ma cenu delat mu benchmarky?

Last observed feature of the solvers will be usability of results in external programs. Solvers can be used independently just for solving given problem but more common use would be using solver as constraint solving subsystem of larger system. This means that solver should be equipped with interfaces to programming languages, provides outputs in machine readable formats or solver itself is a library which is directly usable in other systems.