Constraint Solvers

First we will remind what a constraint solver is. Constraint solver is a system which takes a description of constraint satisfaction problem as input and produces solution which fits to all of the constraints. In this chapter we will describe several solvers by subjective criterions. We will mention short history of the solver and then we will try to discuss the usability and easiness of learning of given solver. As stated in Methodology chapter we will use the point of view of user experienced in given programming language (if the solver use any) but inexperienced in using of the solver. The solver is for us a black-boxed tool and we will not study the inner implementation of the solvers even it could be possible in case of open source solvers.

# Minion

Minion solver is not a programming language library and is not configured by input in some programming language either. Minion provides limited set of constraints and as input uses its own input file format. The input file is divided into four parts. First part defines variables. User can use Boolean variables, bounds (intervals with stored upper and lower limit) and discrete domain (as a range or as a sparse list of values). The variables can form a multidimensional array. Second part is optional. User can define the tuples used in the table constraint. For example let have a variable q[3] {0..5} which means “vector of three numbers in domain 0..5”. The expected value for q is one of the following tuples: (1,0,1), (0,1,1,), (2,5,3). The constraint table enforces that q can have only given values. If the tuple is used more than once it is better to define a named tuple instead of typing it in all occurrences and then use this name. If there is not needed such named tuple this section can be avoided. Third section defines constraints on the variables. There are no control structures known from higher programming languages. Also it is not possible to use nested constraints like:

constraint1(constraint2(x)).

We have to use auxiliary variable to achieve this state:

a = constraint2(x)

constraint1(a)

Unfortunately this means that the count of auxiliary variables grows really fast because there have to be introduced such variable almost for each relation of two constrained variables. This also means that the problem cannot be easily scaled. If we have input file for the 8-queens problem we cannot easily change it to 9-queens problem only by changing one constant. Second big problem could be the given set of constraints. There is constraint for “the sum of given vector is greater or equal than given number” as well as the constraint “the sum of given vector is less or equal than given number”. However there is not the constraint “the sum of given vector is equal than given number” and we can achieve this constraint only by combining of the weak inequality constraints.

## Tailor

As described in previous paragraph Minion input is hard to scale and even hard to write. User needs to specify the problem in atomic terms and relations. This disturbs the user which is no longer focusing on his problem which is to be solved but instead is wasting time with defining new auxiliary variables etc. Fortunately there exists the Tailor tool. Tailor is an application which compiles Essence programs to minion input files. Since it is written in Java it is multiplatform and without changes can be run under MS Windows as well as under Linux etc. Tailor uses as input Essence program and converts it into minion input. User can save the output or run the solver from the Taylor interface (Tailor will run Minion in the background and shows the output). For better scaling and separation of data and code Tailor takes two files as input. First is the definition of constraint program which can delegate some variables to the second file which specifies these variables. For example if we want to solve 4-,5-,6-queens problems we can have only one n-queens source code and change only the data file with the constant n=4,5,6.

If there would be no such system as Tailor the usability of the Minion solver would be very low. The input format itself is very hostile for newcomers and for real usage the system is too rough. But since Tailor use as input language Essence it is more user-friendly.

# Gecode

Gecode je C++ open source knihovna pro řešení úloh s omezujícími podmínkami. Donedávna byla tato knihovna velmi špatně zdokumentovaná, což tvořilo velký problém uživatelům, kteří s knihovnou začínali. Gecode bylo nedávno přepsáno do verze 3 a s touto verzí přišla i o mnoho lepší dokumentace. K dispozici je nyní přibližně stostránkový manuál, který postupně uživatele provede celým používáním knihovny. Před vydáním této dokumentace byl uživatel odkázán pouze na dokumentaci ve stylu Doxygen, která nebyla příliš návodná.

# Choco

Choco je open source knihovna pro řešení úloh s omezujícími podmínkami. Jedná se o knihovnu v programovacím jazyce Java, která je dobře zdokumentovaná a poskytuje široké možnosti modelování problému. Díky tomu, že je distribuovaná jako knihovna, není problém použít Choco jako subsystém existujícího programu pro řešení omezujících podmínek. Knihovna je nyní v hlavní verzi 2, má za sebou tedy již nějakou dobu vývoje. Verze 1 a 2 nejsou zcela kompatibilní. Základní koncepty zůstaly stejné, ale ve verzi 2 způsob popisu problému doznal několika významných změn. Samotný problém je rozdělen na dvě části. Model, který popisuje proměnné a podmínky na nich a řešič, který na vstupu dostane model a poskytuje řešení. V řešiči je také možné určit strategii užitou při ohodnocování proměnných modelu a tím přizpůsobit způsob řešení konkrétnímu problému.

Choco umožňuje použít běžnou sadu podmínek – binární relace, aritmetické podmínky, podmínky pro plánování, množinové podmínky, a další. V případě, že podmínky k dispozici nestačí k popisu úlohy, je možné přidat vlastní podmínky do modelu. Toto umožňuje v Choco modelovat i poměrně velké úlohy bez větších problémů. Autor se tedy může zaměřit na samotný popis problému v omezujících podmínkách a nemusí brát velké ohledy na systém, který je řeší. Toto se projevilo i u modelových úloh, definovaných v kapitole benchmarky. Při jejich implementaci v Choco nenastaly žádné neočekávané těžkosti, které by komplikovaly jejich implementaci.