Course Assignment: Implementing a CSP Solver

Martin Fränzle Andreas Eggers Tino Teige

May 30, 2007

1 Introduction

This assignment deals with the implementation of a solver for *constraint satisfaction problems* (CSPs). Constraint satisfaction is one of the most important and well studied areas in artificial intelligence. There is an extensive literature concerning this issue (for a compendium cf. [RvBW06] A CSP is a high level description of a problem, its range of application contains Planning and Scheduling Problems, Network Management and Configuration, Database Systems, Molecular Biology, Electrical Engineering and many more.

The CSP model is represented by a set of variables and their domains as well as by constraints specifying the relations between the variables. The variable domain is not limited to be finite or bounded but may be continuous. Moreover, constraints can involve arbitrary arithmetics. Therefore, a CSP is a very useful and rich tool for describing (real-world) models and problems, and strongly generalizes the well-known NP-complete Boolean satisfiability problem (SAT).

In this assignment we just consider a very restricted class of CSP problems over finite domains involving offset arithmetic. Nevertheless, in this restricted class problems can be encoded more conveniently than in SAT.

In the lectures you got to know the DPLL algorithm for solving propositional SAT problems. Modern DPLL SAT solvers are very efficient and are able to handle huge formulas due to recent enhancements like conflict-driven learning, non-chronological backtracking, and lazy clause evaluation.

Objective. Your task is to develop a simple branch-and-prune CSP solver based on the DPLL backtrack search for a more general class of formulas, where the domain of a variable is no longer two-valued. This means that your implementation should provide an extended DPLL decision mechanism, backtracking, and the deduction process.

2 Constraint satisfaction problems

The classic definition of a *constraint satisfaction problem* is as follows:

Definition 1 (CSP) *A* CSP *is a triple* $\mathcal{P} = \langle X, D, C\rangle$ *where*

- $X = \langle x_1, \ldots, x_n\rangle$ *is an n-tuple of variables,*
- $D = \langle D_1, \ldots, D_n\rangle$ *is a corresponding n-tuple of domains s.t.* $x_i \in D_i, i \in \{1, \ldots, n\}$*, and*
- $C = \langle c_1, \ldots, c_t\rangle$ *is a t-tuple of constraints.*

A constraint c_j *is a pair* $\langle R_{S_j}, S_j\rangle$ *where* S_j *is an* n'*-tuple of variables in* X*, where* $n' \leq n$*, and* R_{S_j} *is a relation on the variables in* S_j*, i.e.* R_{S_j} *is a subset of the Cartesian product of the domains of the variables in* S_j*.*

A solution *of a CSP* $\mathcal{P}$ *is an n-tuple* $A = \langle a_1, \ldots, a_n\rangle$ *where* $a_i \in D_i$ *and each constraint* c_j *is satisfied, i.e. A projected onto* S_j *is an element of* R_{S_j}*. Note that the relation* R_{S_j} *can be given explicitly or implicitly by a predicate. A CSP* $\mathcal{P}$ *is called* satisfiable *if a solution of* $\mathcal{P}$ *exists, otherwise* $\mathcal{P}$ *is* unsatisfiable*.*

Example 1 *Consider the trivial CSP* $\mathcal{P} = \langle\langle x_1, x_2\rangle, \langle D_1, D_2\rangle, C\rangle$ *where* x_1, x_2 *are reals,* $D_1 = [0,3], D_2 = [2,5]$ *and* $C = \langle\langle\{\langle w_1, w_2\rangle : w_1 = w_2\}, \langle x_1, x_2\rangle\rangle\rangle$. *The only constraint states that* $x_1 = x_2$ *has to hold. A solution of* $\mathcal{P}$*, therefore, is e.g.* $\langle 2.7, 2.7\rangle$. *However, if* $D_2 = [4,5]$ *then* $\mathcal{P}$ *is unsatisfiable since the domains of* x_1 *and* x_2 *are disjoint, i.e.* $D_1 \cap D_2 = \emptyset$.

2.1 Simple CSPs

Your CSP solver should address just a subclass of the general CSP definition. We restrict the domains D_i to be finite and bounded. The domains D_i are given by the interval $[lb_i, ub_i] \subset \mathbb{Z}$ where $lb_i, ub_i \in \mathbb{Z}$ are whole numbers (integers). Furthermore, we just consider a special form of constraints. Therefore, we need some definitions.

Definition 2 (simple bound, simple constraint) *Let* x *and* y *be two integer variables and* $k \in \mathbb{Z}$ *be a whole number, then*

$$x \geq y + k$$

is called a simple bound.

A constraint $c = \langle R_S, S\rangle$ *is called* simple *if and only if* R_S *can be represented by a disjunction* $\mathcal{D}_{R_S}$ *of simple bounds*[1]. *For the sake of simplicity we write* $\mathcal{D}_{R_S}$ *to denote the constraint* c.

Note the following observations, where x and y are integer variables, $\mathbf{z} \in [0,0]$ is a dedicated integer variable representing the value 0, and $k, k', l \in \mathbb{Z}$.

$$\begin{array}{rcll}
x \geq y & \equiv & x \geq y + 0 & \\
x \geq y - k & \equiv & x \geq y + k' & \text{where } k' = -1 \cdot k \\
x \geq k & \equiv & x \geq \mathbf{z} + k & \\
k \geq l & \equiv & \mathbf{z} \geq \mathbf{z} + k' & \text{where } k' = l - k \\
\\
x \leq y + k & \equiv & y \geq x - k & \\
x \leq y & \equiv & y \geq x & \\
x \leq y - k & \equiv & y \geq x + k & \\
x \leq k & \equiv & \mathbf{z} \geq x - k & \\
k \leq l & \equiv & l \geq k & \\
\\
x > y + k & \equiv & x \geq y + k' & \text{where } k' = k + 1 \\
x > y & \equiv & x \geq y + 1 & \\
x > y - k & \equiv & x \geq y - k' & \text{where } k' = k + 1 \\
x > k & \equiv & x \geq k' & \text{where } k' = k + 1 \\
k > l & \equiv & k \geq k' & \text{where } k' = l + 1 \\
\\
x < y + k & \equiv & y > x - k & \\
x < y & \equiv & y > x & \\
x < y - k & \equiv & y > x + k & \\
x < k & \equiv & x \leq k' & \text{where } k' = k - 1 \\
k < l & \equiv & l > k &
\end{array}$$

Example 2 *Let* $c = \langle R_S, S = \langle x_1, x_2\rangle\rangle$ *where* $x_i \in D_i = [-1,2] \subset \mathbb{Z}$ *for* $i \in \{1,2\}$, *and*

$$\begin{array}{rl}
R_S = \{ & (-1,-1), (-1,0), (-1,1), (-1,2), \\
& (0,2), (1,2), (2,2), \\
& (2,-1) \ \}.
\end{array}$$

The pairs $(-1, v_{x_2})$ *for* $v_{x_2} \in D_2$ *can be encoded as solutions in* $D_1 \times D_2 = [-1,2] \times [-1,2]$ *of the predicate* $x_1 \leq -1$. *The same holds for the pairs* $(v_{x_1}, 2)$ *for* $v_{x_1} \in D_1$ *as solutions of* $x_2 \geq 2$. *The pair* $(2,-1)$ *is the only solution in* $D_1 \times D_2$ *of the predicate* $x_1 \geq x_2 + 3$. *Thus, the relation* R_S

[1] The definition of satisfaction of simple bounds and disjunctions thereof is standard.

```
DECL
   x_0  0  0;
   x_1  3  6;
   x_2  -6  4;
   x_3  -2  5;

FORMULA
   x_1 >= x_0 + -1 v x_0 >= x_1 + 3 v x_2 >= x_1 + 3 v x_3 >= x_2 + -1;
   x_1 >= x_0 + -4 v x_0 >= x_2 + 6 v x_3 >= x_2 + 4 v -10 >= 4;
   x_1 >= x_0 + -2 v x_0 >= x_3 + 6 v x_3 >= x_1 + 0 v 4 >= -10;
```

Figure 1: Example of the input format (corresponds to the CSP from example 3)

contains exactly all solutions in $D_1 \times D_2$ of the predicate $P = (x_1 \leq -1 \vee x_2 \geq 2 \vee x_1 \geq x_2 + 3)$. By the equivalence relations above P can be translated into the simple constraint

$$c = (\mathbf{z} \geq x_1 + 1 \vee x_2 \geq \mathbf{z} + 2 \vee x_1 \geq x_2 + 3) \equiv P$$

where $\mathbf{z} \in [0, 0]$ is a (fresh) dedicated variable representing 0.

To sum up, we define the restricted CSP.

Definition 3 (simple CSP) *A CSP $\mathcal{P} = \langle X, D, C \rangle$ is called* simple *iff*

- *$\forall D_i \in D \; \exists lb_i, ub_i \in \mathbb{Z}$ (with $lb_i \leq ub_i$) : $D_i = [lb_i, ub_i] \subset \mathbb{Z}$, and*
- *each constraint $c \in C$ is simple.*

Example 3 *Given the simple CSP $\mathcal{P} = \langle X, D, C \rangle$ where $X = \langle x_0, x_1, x_2, x_3 \rangle$ and $D_0 = [0, 0], D_1 = [3, 6], D_2 = [-6, 4], D_3 = [-2, 5]$ are the domains of the variables. C contains the following simple constraints:*

$$\begin{array}{l} (x_1 \geq x_0 + (-1) \;\vee\; x_0 \geq x_1 + 3 \;\vee\; x_2 \geq x_1 + 3 \;\vee\; x_3 \geq x_2 + (-1)), \\ (x_1 \geq x_0 + (-4) \;\vee\; x_0 \geq x_2 + 6 \;\vee\; x_3 \geq x_2 + 4 \;\vee\; -10 \geq 4), \\ (x_1 \geq x_0 + (-2) \;\vee\; x_0 \geq x_3 + 6 \;\vee\; x_3 \geq x_1 + 0 \;\vee 4 \geq -10). \end{array}$$

A solution of the CSP $\mathcal{P}$ is e.g. $x_0 = 0$, $x_1 = 4$, $x_2 = -1$, and $x_3 = 1$, since $x_3 \geq x_2 + (-1)$ is satisfied in the first constraint, $x_1 \geq x_0 + (-4)$ in the second, and, trivially, $4 \geq -10$ in the third.

2.2 Input format of the tool

Your tool must accept input files of the following syntax:

$$\begin{array}{rcl} \mathit{simple\_CSP} & ::= & \texttt{DECL}\ \mathit{var\_decl} \\ & & \texttt{FORMULA}\ \mathit{formula} \\ \mathit{var\_decl} & ::= & \{\mathit{variable}\ \ \mathit{lb}\ \ \mathit{ub}\ \texttt{;}\}^* \\ \mathit{formula} & ::= & \{\mathit{constraint}\ \texttt{;}\}^* \\ \mathit{constraint} & ::= & \{\mathit{simple\_bound}\ \texttt{v}\}^*\ \ \mathit{simple\_bound} \\ \mathit{simple\_bound} & ::= & \mathit{variable}\ \texttt{>=}\ \mathit{variable}\ \texttt{+}\ \mathit{integer} \\ \mathit{variable} & ::= & \texttt{x\_}\mathit{nat\_number} \end{array}$$

where $nat\_number \in \mathbb{N}$, and $lb, ub, integer \in \mathbb{Z}$.

The semantics of these constraint formula is standard, as explained in Fig. 1.

3 Algorithm for solving simple CSPs

The backtrack algorithm as mainly described in subsection 3.1 for solving simple CSPs $\mathcal{P} = \langle X, D, C\rangle$ manipulates *interval valuations* (or *interval assignments*) $\rho : X \to \mathbb{I}$, $\emptyset \notin \mathbb{I}$, mapping a variable $x_i \in X$ to a non-empty interval $\rho(x_i) \in \mathbb{I}$ with $\rho(x_i) \subseteq D_i \subset \mathbb{Z}$. Let ρ and ρ' be two interval valuations. If $\forall x \in X : \rho'(x) \subseteq \rho(x)$ then ρ' is called *refinement* of ρ.

The initial interval valuation ρ_0 maps each variable x_i to its initial interval D_i, i.e. $\forall x_i \in X : \rho_0(x_i) = D_i$. Therefore, we have to define the interpretation of simple bounds over interval valuations.

Definition 4 (Semantics of simple bounds over interval valuations) *Let ρ be an interval valuation. Then a simple bound $x \geq y + k$ is*

- true *under ρ iff*
$$\min \rho(x) \geq \max \rho(y) + k,$$
- false *under ρ iff*
$$\max \rho(x) < \min \rho(y) + k,$$
and
- inconclusive *under ρ otherwise.*

Example 4 *Let ρ be defined by $\rho(x) = [-2, 4]$ and $\rho(y) = [2, 10]$. Then*

- *$x \geq y + (-15)$ is true under ρ, since*
$$\min \rho(x) = \mathbf{-2} \;\geq\; \max \rho(y) + (-15) = 10 - 15 = \mathbf{-5},$$
- *$x \geq y + 15$ is false under ρ, since*
$$\max \rho(x) = \mathbf{4} \;<\; \min \rho(y) + 15 = 2 + 15 = \mathbf{17},$$
and
- *$x \geq y + 0$ is inconclusive under ρ, since both*
$$\min \rho(x) = \mathbf{-2} \;\geq\; \max \rho(y) + 0 = \mathbf{10},$$
and
$$\max \rho(x) = \mathbf{4} \;<\; \min \rho(y) + 0 = \mathbf{2},$$
do not hold.

The definition of satisfaction of simple constraints over interval assignments, then again, is standard. A simple constraint is *satisfied* by or *true* under an interval valuation ρ iff at least one of its simple bounds is true under ρ. A simple constraint is false under ρ iff all its simple bounds are false under ρ. If a simple constraint c is neither true nor false under ρ, we say that c is inconclusive under ρ.

The next proposition states that interval satisfaction of a CSP $\mathcal{P}$ is a sufficient condition for satisfiability of $\mathcal{P}$.

Proposition 1 *Let ρ be an interval valuation. Then, if all simple constraints of a CSP $\mathcal{P}$ are satisfied by ρ then there exists a (point) solution of $\mathcal{P}$ and, hence, $\mathcal{P}$ is satisfiable.*

3.1 Basic backtrack algorithm

Given a simple CSP $\mathcal{P} = \langle X, D, C\rangle$. In order to decide whether $\mathcal{P}$ is satisfiable or unsatisfiable, the base algorithm $\mathcal{A}$ consecutively performs two steps, a *consistency checking step* and a *decision step* (by splitting intervals), until inconsistency (a constraint becomes false) or satisfiability (all constraints are satisfied) are detected. In case of inconsistency, a backtrack step is applied. If backtracking is not feasible then we stop with result "$\mathcal{P}$ is unsatisfiable". If satisfiability is proven, we return "$\mathcal{P}$ is satisfiable".

Algorithm $\mathcal{A}$

1. **Consistency check:** For each simple constraint $c \in C$, the algorithm computes the truth value of c under the current interval valuation ρ. If a false constraint is found go to 2. In the case that all constraints are true under ρ, then stop with the result "$\mathcal{P}$ is satisfiable". Otherwise go to 3.

2. **Backtrack:** Retrieve the interval valuation ρ which was valid before the youngest non-revoked decision step was performed. Assert the stored alternative interval *(cf. 3)* of that decision, update ρ correspondingly, and go to 1.

   If no such decision exists, return "$\mathcal{P}$ is unsatisfiable" and stop.

3. **Decision:** Select a variable $x \in X$ s.t. $|\rho(x)| > 1$ and split the interval $\rho(x)$ into two disjoint intervals $\rho'(x)$ and $\rho''(x)$ (with $\rho'(x) \cup \rho''(x) = \rho(x)$). Opt for one of the intervals and update the current interval valuation correspondingly. Store the other interval as the alternative, and go to 1.

We say that each decision opens a new decision level, where initially solving starts on decision level 0. Each decision increments the decision level by 1.

Example 5 *Given the simple CSP $\mathcal{P} = \langle X, D, C\rangle$ where $X = \langle x_0, x_1, x_2\rangle$ and $D_0 = [0,0], D_1 = [1,6], D_2 = [-2,3]$. C contains the following simple constraints:*

$$c_1 = (x_1 \geq x_0 + 5 \ \vee \ x_2 \geq x_1 + (-5)),$$
$$c_2 = (x_0 \geq x_2 + 2 \ \vee \ x_1 \geq x_2 + 4).$$

Algorithm $\mathcal{A}$ starts solving with the initial interval valuation ρ_0 at step 1 on decision level 0. All simple bounds are inconclusive under ρ_0, thus, no constraint is false under ρ_0. $\mathcal{A}$ performs a decision (step 3). The variable x_0 cannot be split since its interval is a point interval. So, we select variable x_1 and split its current interval $\rho_0(x_1) = [1,6]$ into $[1,3]$ and $[4,6]$. We decide for $[1,3]$, update the interval valuation to ρ_1, where $\rho_1(x_1) = [1,3]$ and $\rho_1(x_i) = \rho_0(x_i)$ for $i \neq 1$, and open decision level 1. The interval $[4,6]$ is stored as the alternative for possible backtracking.

Again $\mathcal{A}$ checks the consistency of $\mathcal{P}$. The simple bound $x_1 \geq x_0 + 5$ in c_1 becomes false under ρ_1 since

$$\max \rho_1(x_1) = \mathbf{3} \ < \ \min \rho_1(x_0) + 5 = 0 + 5 = \mathbf{5},$$

while $x_2 \geq x_1 + (-5)$ becomes true since

$$\min \rho_1(x_2) = -\mathbf{2} \ \geq \ \max \rho_1(x_1) + (-5) = 3 - 5 = -\mathbf{2}.$$

Thus, the simple constraint c_1 evaluates to true. Both bounds in the second constraint c_2 are still inconclusive under ρ_1.

We go to step 3 and split the interval $\rho_1(x_2) = [-2,3]$ of x_2 into $[-2,0]$ and $[1,3]$. We decide for the upper part $[1,3]$, store $[-2,0]$ as the backtrack alternative, and open decision level 2, where ρ_2 is the updated interval valuation. Then, the constraint c_2 becomes false, since

$$\max \rho_2(x_0) = \mathbf{0} \ < \ \min \rho_2(x_2) + 2 = 1 + 2 = \mathbf{3}$$

and

$$\max \rho_2(x_1) = \mathbf{3} \;<\; \min \rho_2(x_2) + 4 = 1 + 4 = \mathbf{5}$$

hold. Therefore, $\mathcal{A}$ backtracks before the last non-revoked decision and re-opens decision level 1 *with the interval valuation ρ_1. The stored alternative interval $[-2, 0]$ for variable x_2 of that decision is asserted. Let ρ_3 be the current interval assignment. The evaluation of constraint c_1 still holds under ρ_3, i.e. c_1 is true under ρ_3, while both simple bounds in c_2 are inconclusive again.*

Consequently, $\mathcal{A}$ opens decision level 2 *and splits an interval: $\rho_3(x_2) = [-2, 0]$ into $[-2, -2]$ and $[-1, 0]$, where now the lower part $[-2, -2]$ is decided which leads to the interval valuation ρ_4. This implies that $x_0 \geq x_2 + 2$ becomes true since*

$$\min \rho_4(x_0) = \mathbf{0} \;\geq\; \max \rho_4(x_2) + 2 = (-2) + 2 = \mathbf{0}.$$

This means that also constraint c_2 is satisfied by ρ_4. Hence, the algorithm $\mathcal{A}$ stops with result "$\mathcal{P}$ is satisfiable".

3.2 Acceleration techniques

As the (finite-domain) constraint satisfaction problem is NP-complete in general, it is not expected that there exists an algorithm (on deterministic machines) which efficiently solves the problem in theory, i.e. with a polynomial time bound. Not surprisingly, the worst-case time complexity of algorithm $\mathcal{A}$ is exponential in the number of variables.

However, e.g., industrial needs motivate investigations of acceleration techniques for the basic backtrack approach. Recent algorithmic enhancements in the SAT and CSP community, like conflict-driven learning and non-chronological backtracking, led to efficient algorithms from a practical point of view.

In the sequel we consider some acceleration techniques where our main focus is on the deduction process while others are just mentioned briefly.

Deduction (a.k.a. constraint propagation). One powerful mechanism is the deduction process. In contrast to pure branching and consistency checking, here we derive new information (e.g., about the interval valuation) from the current state of the proof search. In our case, if all but one simple bound in a simple constraint c are false under the current interval valuation ρ then we can deduce that the remaining (still inconclusive) simple bound has to hold. Otherwise, the entire constraint becomes false which is a dead end in our proof search and implies a backtrack. We call such a constraint c *unit*.

More formally, let $c = (sb_1 \vee \ldots \vee sb_i = x \geq y + k \vee \ldots \vee sb_n) \in C$ be a simple constraint in a constraint system C and ρ be an interval valuation. Moreover, let all simple bounds $sb_j \in c$ for $j \neq i$ be false under ρ, and sb_i be inconclusive under ρ. To satisfy C under a refinement ρ' of ρ we have to satisfy the constraint c and thus also sb_i. Slightly more formally:

$$C \text{ is true under } \rho' \;\text{ only if }\; c \text{ is true under } \rho' \;\text{ iff }\; sb_i \text{ is true under } \rho'.$$

So, each tuple t of values for the variables in ρ not satisfying the simple bound sb_i does not satisfy the constraint system C, i.e. t projected on x and y is not a solution of sb_i. In order to exclude such tuples we try to deduce new intervals for the variables x and y from ρ and sb_i as follows:

- From $x \geq y + k$ it follows that $x \geq (\min \rho(y) + k) =: lb$. So, we can potentially deduce a new lower bound for the interval of x, i.e. $\rho'(x) = \rho(x) \cap [lb, +\infty)$.
- From $x \geq y + k \equiv y \leq x - k$ it follows that $y \leq (\max \rho(x) - k) =: ub$. So, we can potentially deduce a new upper bound for the interval of y, i.e. $\rho'(y) = \rho(y) \cap (-\infty, ub]$.

For all other variables $z \neq x, y : \rho'(z) = \rho(z)$. Note that it is not guaranteed in general that the deduced lower or upper interval bounds for x or y are actually tighter. However, ρ' is always a refinement of ρ, since $\rho'(x) \subseteq \rho(x)$ and $\rho'(y) \subseteq \rho(y)$.

Example 6 *Let* $c = (x \geq y + 0 \vee y \geq z + 4)$ *be a simple constraint, and* ρ *given by* $\rho(x) = [0, 1], \rho(y) = [4, 20], \rho(z) = [10, 15]$ *be an interval valuation. It is easy to see that* $x \geq y + 0$ *is false and* $y \geq z + 4$ *is inconclusive under* ρ*. So, we can deduce new bounds from* $y \geq z + 4$*:*

- $y \geq \min \rho(z) + 4 = 10 + 4 = 14$.
- $z \leq \max \rho(y) - 4 = 20 - 4 = 16$.

Hence, $\rho'(y) = \rho(y) \cap [14, +\infty) = [4, 20] \cap [14, +\infty) = [14, 20]$ *and* $\rho'(z) = \rho(z) \cap (-\infty, 16] = [10, 15] \cap (-\infty, 16] = [10, 15]$*. We see that we narrowed down the interval for* y *enormously while the interval of* z *remains untouched.*

We enrich algorithm $\mathcal{A}$ of subsection 3.1 by a deduction process.

Algorithm $\mathcal{B}$ works as algorithm $\mathcal{A}$ where step 1 is enriched as follows:

1. **Consistency check & Deduction:** For each simple constraint $c \in C$ the algorithm computes the truth value of c under the current interval valuation ρ. If a false constraint is found go to 2. In the case that all constraints are true under ρ, then stop with result "$\mathcal{P}$ is satisfiable". If one constraint is unit then deduce new intervals and update ρ correspondingly. If the narrowing was successful then go to 1, otherwise go to 3.

Note that deducing new interval valuations does not increment the decision level, which is only done by step 3.

Lazy clause evaluation. To accelerate step 1 of algorithms $\mathcal{A}$ or $\mathcal{B}$ we just need to visit constraints containing variables for which the interval was narrowed down instantaneously before. The truth value of all other constraints cannot change. Moreover, it is not necessary to visit already satisfied constraints. Another very powerful technique is the *two-watch-literal scheme*, where only two bounds of a constraint are evaluated.

Conflict-driven learning & non-chronological backtracking. If a conflict in the proof search was detected then a (minimal) reason for this conflict can be determined (with some additional data-structures). Such a reason contains information about the intervals (or rather interval borders) of some variables contributing to the conflict. In order to avoid visiting a conflict with the same reasons in another part of the search space, the negation of this reason can be added/ learned as a further constraint (called *conflict clause* in the SAT community).

As witnessed by the reason of a conflict it might happen that a few of the last decisions have no impact on the conflict. Therefore, we are able to skip backtracking to these decision levels and jump back to the second highest decision level contributing to the conflict. Then, the learned constraint will prevent us from visiting the conflict again.

Both techniques allow to achieve enormous performance gains in SAT and CSP solving.

Decision heuristics. Since our problem lies in NP, where a solution can be non-deterministically guessed efficiently, heuristics for the choice of a variable to be split play an important role and can lead to speed-ups of multiple orders of magnitude (for special cases). One of the famous decision heuristics is the so called *variable state independent decaying sum* (VSIDS) decision heuristics. Here, the activities of the variables during the solving process are taken into account.

4 Assignment

The assignment is subdivided into a theoretical and practical part. The theoretical considerations should demonstrate your understanding of the theoretical basis of the algorithm, while the practical part deals with the development of a CSP solver tool implementing the algorithm.

Note that there are *optional* exercises which are not required to solve. However, fulfillment of these optional tasks leads to **bonus points**.

4.1 Exercises

1. Theoretical consideration

   (a) Why does proposition 1 hold? How can a solution be extracted from ρ? Give a comprehensible explanation. (A formal proof is not necessary.)

   (b) Is algorithm $\mathcal{A}$ *sound* and *complete*, i.e. does $\mathcal{A}$ stop on each simple CSP with the correct result? Give reasons for your answer.

   (c) Assume that we generalize the definition of a simple CSP s.t. the domains of the variables are (possibly) *infinite*, but still bounded, e.g. intervals over the reals, and simple bounds are of the form $x \sim y + r$ with $\sim \in \{\geq, >\}$ and $r \in \mathbb{R}$. Let $\mathcal{P}$ be such a generalized simple CSP.

      i. Does algorithm $\mathcal{A}$ work correctly on $\mathcal{P}$, i.e. is the result correct? If this is not the case, can $\mathcal{A}$ be adapted? Give reasons for your answer.

      ii. Does algorithm $\mathcal{A}$ terminate on $\mathcal{P}$? If this is not the case, can $\mathcal{A}$ be adapted? Give reasons for your answer.

      Remark: we consider the algorithm here in theory and not its behavior on a resource-limited machine.

   (d) **Optional:** Think about possible decision heuristics: Which heuristics –in your opinion– for a variable choice, for interval splitting and interval deciding make sense in this framework and why?

   (e) **Optional:** Is it possible to integrate conflict-driven learning and non-chronological backtracking into algorithm $\mathcal{B}$? Briefly sketch your ideas.

2. Practical work

   For the implementation part you can use your favorite programming language.

   (a) Think about suitable data structures for the internal representation of variables, simple bounds, simple constraints, interval valuations, etc. At the same same time, reflect on the aim of the data structures, e.g. use a stack for storing interval valuations or rather intervals in order to easily recover interval valuations when backtracking. Implement the data structures.

   (b) Implement a parser for reading in the input file to the data structure of your solver. Remark: Your parser can simply do that on-the-fly.

   (c) Implement algorithm $\mathcal{A}$ (subsection 3.1). If the input CSP is satisfiable then $\mathcal{A}$ must output a solution.

   (d) Implement algorithm $\mathcal{B}$ (subsection 3.2). If the input CSP is satisfiable then $\mathcal{B}$ must output a solution. Therefore, extend the solver from exercise 2c.

   (e) Test both tools (from exercises 2c and 2d) on a sufficient number of input files. You can also try to find suitable application examples, e.g. from scheduling or other planning problems and encode these apropriately. Measure and compare the runtimes of both CSP solvers on all your benchmarks (also these from exercise 2e). Do the runtimes differ from each other and, if so, why? For a fair comparison all benchmarks should be performed on the same machine.

   (f) **Optional:** Enhance algorithm $\mathcal{B}$ by supporting some kind of

      - lazy clause evaluation,
      - decision heuristics

      mentioned in subsection 3.2. Does this actually improve the runtimes on your benchmarks over algorithm $\mathcal{A}$ and $\mathcal{B}$?

5 Report requirements

Your work on this assignment must be documented by a report. The report should comprise the following:

- A short introduction and problem description.
- The solutions to the exercises.
- Implementation details should not be part of the report. (If desired or necessary add such details to the appendix.) However, all your implemented solutions have to be described briefly in your own words and possibly supported by some figures, in particular the realized data structures and main procedures of the algorithm.
- Encountered problems and your solutions.
- A conclusion in which you should summarize your findings obtained on this project.

Report Form

The report must be handed in *on paper*. In case this is impractical, you should ask Michael for permission for an electronic only submission. Deadlines apply in the same way.

Further, the various input files, output files, the well-documented source files of the tool, etc. must be packed into a single *zip* or *tar.gz* file for each report and sent to MF (`Fraenzle@informatik.uni-oldenburg.de`) by email. After unpacking, the various files must be clearly identifiable.

The report must have a front page identifying the course, the assignment and the participating students.

The report on the assignment is expected to be around 10 pages and should not exceed 15 pages. To this you may add appendices.

General

The report for this assignment must be handed in at the *IMM reception, building 321* no later than **Friday, July 11, 2008 at 12.00**. This is a *hard deadline* — no reports handed in later will be considered. Also the files must be sent before the deadline.

Please note:

- The assessment will be based upon the presentation of your work in the reports.
- The reports *must be signed* by all participants. Unless you state otherwise, the signatures are understood to confirm that all participants agree to have contributed equally to the project.
- Any collaboration with other groups on smaller parts of the assignments must be declared and clearly identified. Collaboration on major parts is not acceptable.

Also note that clarifications, FAQs, and practical details may be put on the *course project page* found via the course home page. You should consult this if you encounter problems.

References

[RvBW06] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming.* Foundations of Artificial Intelligence. Elsevier, Amsterdam, 2006.