# A General Purpose Local Search Solver

October 21, 2015

## Contents

# 1 Introduction

## 1.1 Mixed Integer Programming

## 1.2 Constraint Programming

## 1.3 Heuristics and Local Search

### 1.3.1 Construction Heuristics

### 1.3.2 Local Search and Neighborhoods

### 1.3.3 Metaheuristics

# 2 Modeling in CBLS

## 2.1 Types of Modeling (not sure this should be here)

## 2.2 Variables

Models contains variables $X$ that is a set of $n$ variables $X = \{x_1, x_2, \ldots, x_n\}$. Each variable $x_i \in X$ has a *domain* $D(x_i) \in D$ where $D$ is an n-tuple of domains $D = \langle D_1, D_2, \ldots, D_n \rangle$ such that $x_i \in D_i$. The variables $x_i \in X$ of the models that will be discussed in this thesis all have their domain restricted toa finite descrete domain $D_i \subseteq \mathbb{Z} \ : \ \forall \ D_i \in D$. The value of a variable $x$ is denoted $V(x)$ and we will denote integer variables as $y_i \in Y \subseteq X$.

## 2.3 Constraints

The variables will be restricted by $C$ that is a set of $m$ constraints $C = \{c_1, c_2, \ldots, c_m\}$. The set of variables to which the constraint $c_j$ **(Drop subscript j?)** applies is called its *scope* and is denoted $X(c_j) = \{c_{i_1}, \ldots, c_{i_{|X(c_j)|}}\}$. The size of a scope $|X(c)|$ is called the *arity* $\alpha(c)$. The constraint $c_j$ is a subset of the cartesian product of the domains of the variables in the scope $X(c_j)$ of $c_j$. $c_j \subseteq D(x_{i_1}) \times D(x_{i_2}) \times \cdots \times D(x_{i_{\alpha(c_j)}})$. The Constraint Satisfaction Problem (CSP) can then be defined as a triple $\mathbb{P} = \langle X, D, C \rangle$. A *solution* to the CSP $\mathbb{P}$ is a n-tuple $\mathcal{T} = \langle \tau_1, \tau_2, \ldots, \tau_n \rangle$ where $\tau_i \in D_i$. The solution is feasible if $\mathcal{T}$ can be projected onto $c_j$ for all $c_j \in C$. **(Er det helt rigtigt?)**
The questions to a CSP could be to report all feasible solutions $sol(P)$, any feasible solution $\mathcal{T}$ or if there exists a solution $\mathcal{T}$ or not.
The CSP $\mathbb{P}$ can be expanded to a Constraint Satisfaction Optimization Problem (CSOP) $\mathbb{P}'$ with an objective function $f(\mathcal{T})$ that evaluates the quality of

the solution $\mathcal{T}$, $\mathbb{P}' = \langle X, D, C, f(\mathcal{T})\rangle$. The task is then to find a solution $\hat{\mathcal{T}}$ that gives minimum or maximum value of $f(\hat{\mathcal{T}})$ depending on the requirements of the problem.

## 2.4 Considered Constriants (Not sure about the title and not finished yet)

While Constraint Programming often offers a wide selection of constraints to use, this work focuses mostly on the constraint **Linear** that is defined by a left hand side, a relation $R$ and a right hand side, which is a bound $b$. The left hand side is a linear function of decision variables multiplied by coefficients. The relation $R$ between left hand side and right hand side is restricted to be one of the six $R \in \{<, \leq, >, \geq, =, \neq\}$
A linear constraint $c$ can be described as:

$$\sum_{x_j \in X(c)} a(x_j) \cdot x_j \;\; R \;\; b(c) \qquad R \in \{<, \leq, >, \geq, =, \neq\} \tag{1}$$

The coefficients $A(c)$ are the coefficients of the variables in the scope of $c$. The decision variables $X(c)$ are the variables that $c$ applies to. The bound $B_c$ is the bound for the left hand side in constraint $c$.
**(Follwing should be a note about CSP is a superset of MIP,IP,BP)**
MIP, IP and BP are restricted to use the linear constraint and the model can be written as:

$$\begin{aligned} \text{Minimize } & \mathbf{c}^T\mathbf{x} \\ \text{Subject to } & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbf{D} \end{aligned} \tag{2}$$

### 2.5 Invariants and One-way Constraints

## 3 Previous Work

### 3.1 Comet

### 3.2 Gecode

### 3.3 LocalSolver

### 3.4 OscaR

## 4 Preprocessing and Simplification

### 4.1 Domain Reduction

### 4.2 Initial Solution

## 5 Structuring Local Search Model

Once an initial solution to the problem has been found by Gecode the model is transformed to create a model better suited for local search. **(talk briefly about DDG and propagation queue (what are they used for))** procedure can be split in several steps before the local search can begin.

1. Define variables by one-way constraints
2. Define invariants for the remaining constraints
3. Create a dependency directed graph for variables and invariants
4. Create propagation queue for variables
5. Initialize the invariants

### 5.1 Simplification

We define as many variables to be one-way constraints as posible, starting with integer variables, such that the search space in local search only consists of binary variables. The following algorithms describe how one-way constraints a create to define a variable $x$.
Let $X$ be a set of variables and $x \in X$. The subset of constraints $C(x) \subseteq C$

is the set of constraints that applies to $x$.

**Algorithm 1:** Defining integer variables by one-way constraints

**input** : A set $X$ of variables **(Sorting order?)**
**output**: A model better suited for local search

```
1  bool change = true
2  while X ≠ ∅ and change do
3      change =false
4      select Variable x from X
5      foreach Constraint c in C(x) do
6          bool flag = canBeMadeOneway(c,x)
7          if flag then
8              makeOneway(c,x)
9              Remove x from X
10             change = true
11             break
12         end
13     end
14 end
```

The algorithm tries to make all integer variables one-way. It uses two other algorithms `canBeMadeOneway(`$c$,$x$`)` and `makeOneway(`$c$,$x$`)`. The first algorithm checks if the **Constraint** $c$ can be used to define **Variable** $x$ and the second algorithm transforms $c$ into a one-way constraint defining $x$. **(Need complexity arguments)**

The coefficient of a variable $x_j$ in constraint $c_i$ is denoted $a_{ij}$. Let $\mathcal{F} = \{f_1, f_2, \ldots, _k$ be the family of objective functions and the coefficient of variable $x_j$ in $f_k$ be $a_{kj}$. **(Maybe call it evalutation functions. Does not make sense since $a_{34}$ referes both to constraint and obj. func)**

**Algorithm 2:** canBeMadeOneway(Constraint c, Variable x)

**input** : Constraint $c$ and Variable $x$
**output**: Boolean

```
1  if c defines a oneway constraint then
2  |  return false
3  end
4  if Number of integer variables not defined > 1 then
5  |  return false
6  end
7  if relation(c) == Equal then
8  |  return true
9  end
10 foreach a in A(f(x)) do
11 |  if A(c,x) · a > 0 then
12 |  |  return false
13 |  end
14 end
15 return true
```

The variables that a constraint $c$ applies to is the scope $V(c)$. The constraints are of the type Linear and a constraint $c$ have a right hand side $B(c)$.

**Algorithm 3:** Make one-way constraint from $c$ defining variable $x$

**input** : Constraint $c$ and Variable $x$
**output**: An Invariant

```
1  int coef = A(c,x)
2  A(c) = A(c)\{A(c,x)}
3  V(c) = V(c)\{x}
4  foreach A(c,v) in A(c) do
5  |  A(c,v) = A(c,v) · -1/coef
6  end
7  int b = B(c)
8  if relation(c) == Equal then
9  |  return Sum(V(c),A(c),b)
10 end
11 else
12 |  Invariant inv =Sum(V(c),A(c),b)
13 |  return Max(inv,b)
14 end
```