A General Purpose Local Search Solver

October 13, 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

2.1 Variables

Models contains variables V that is a n-tuple of variables $V = \langle v_1, v_2, \ldots, v_n \rangle$. Each variable $v_i \in V$ has a domain $D(v_i) \in D$ where D is an n-tuple of domains $D = \langle D_1, D_2, \ldots, D_n \rangle$ such that $v_i \in D_i$. The variables $v_i \in V$ of the models that will be discussed in this paper all have their domain restricted to $D_i \subseteq \mathbb{Z} \ : \ \forall \ D_i \in D$ **(Does not look quite nice)**. Hence the all the variables are discrete and can be incremented in small steps.
From now on $y_i \in Y \subseteq V$ will denote integer variable while $x_i \in X \subseteq V$ denotes binary variables.

2.2 Constraints

The variables will be restricted by C that is a m-tuple of constraints $C = \langle c_1, c_2, \ldots, c_m \rangle$. The set of variables to which the constraint c_j applies is called its scope and is denoted $V(c_j)$ or ($X(c_j)$ and $Y(c_j)$ for the binary and integer variables respectivly). Each $c_j \in C$ is a pair $\langle R_{V(c_j)}, V(c_j) \rangle$ where $R_{V(c_j)}$ is a subset of the cartesian product of the domains of the variables in $V(c_j)$ also called the relation on c_j.
The Constraint Satisfaction Problem (CSP) can then be defined as a triple $\mathbb{P} = \langle V, D, C \rangle$. A solution to the CSP $\mathbb{P}$ is a n-tuple $A = \langle a_1, a_2, \ldots, a_n \rangle$ where $a_i \in D_i$. The solution is feasible if the projection of A onto $V(c_j)$ is included in $R_{V(c_j)}$ for all $c_j \in C$.
The solution of interest could be all feasible solutions $sol(P)$, any feasible solution S or if there exists a solution or not.
The CSP can be expanded to a Constraint Satisfaction Optimization Problem (CSOP) with an objective function $f(S)$ that evaluate the quality of the

solution S. The task is then to find a solution $\hat{S}$ that gives minimum or maximum value of $f(\hat{S})$ depending on the requirements of the problem.

While constraint programming often offers a wide selection of constraints to use, this thesis focus mostly on the constraint Linear that is defined by a left hand side, a relation and a right hand side, which is a constant bound b. The left hand side is a linear function of decision variables multiplied with their coefficient. The relation between left hand side and right hand side is restricted to be one of the six: less ($<$), less or equal ($\leq$), greater($>$), greater or equal($\geq$), equal ($=$), and disequal ($\neq$). **(Ikke særlig pænt, men ved ikke hvordan jeg skal beskrive det ellers (Gecode har seks, MIP og IP har kun 3))**
A linear constraint c can be described as:

$$\sum A(c) \cdot V(c) \lesseqgtr B_c \tag{1}$$

The coefficients $A(c)$ are the coefficients of the variables in the scope of c. The decision variables $V(c)$ are the variables that c applies to. The bound B_c is the bound for the left hand side in constraint c.
MIP, IP and BP are restricted to use the linear constraint and the model is often written as:

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

2.3 Invariants and One-way Constraints

2.4 Types of Modeling (not sure this should be here)

The choice of modeling approach should depend on the nature of the problem. Constraint programming (CP) uses constraints to formulate a constraint satisfaction problem (CSP) and often provides short and natural way to describe a problem. Constraint programming try to find feasible solutions to a CSP hence it does not differentiate the solutions that might exist. One can work around that by creating a function that evaluates a solution by a value. Then add the constraint that the function must give a better value than last time.
(Jeg ved ikke om jeg skal definere IP her eller tidligere (og hvor meget der skal skrive))
Mixed integer programming (MIP) and Integer programming (IP) are more

restricted in the choice of constraints. They can only use linear constraints hence the models often tend to be longer in order to formulate a problem. Unlike constraint programming both mixed integer programming and integer programming has an objective function that rates the quality of a solution. Binary programming (BP) models the same way as MIP and IP but restricts the variables from integer to binary.

3 Different Solvers (properly not the best name)

3.1 Comet

3.2 Gecode

3.3 LocalSolver

3.4 OscaR

3.5 (This solver) Constraint Based Local Search with Limitations

4 Preprocessing and Simplification

4.1 Gecode Engine

4.1.1 Relaxation

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. The procedure can be split in several steps before the local search can begin.

1. Try to define integer variables by one-way constraints
2. Define invariants for the constraints
3. Create a dependency directed graph for variables and invariants
4. Create propagation queue for variables

5. Initialize the invariants
6. Initialize the constraints
7. Initialize the objective function

5.1 Simplification

We want all the integer variables to be defined by one-way constraints such that the search space in local search only consists of binary variables. The following algorithms describe how integer variables get defined by one-way constraints.
let Y be a list of integer variables and $y \in Y$. The subset of constraints $y(c) \subseteq C$ is the set of constraints where integer variable y has a non zero coefficient.

Algorithm 1: Defining integer variables by one-way constraints

input : A List Y of integer variables

```
bool change = true
while Y ≠ ∅ and change do
    change = false
    Variable y = next Variable in Y
    foreach Constraint c in y(c) do
        bool flag = canBeMadeOneway(c,y)
        if flag then
            makeOneway(c,y)
            Remove y from Y
            change = true
            break
        end
    end
end
```

The algorithm try to make all integer variables one-way. It uses two other algorithms `canBeMadeOneway(`c,y`)`2 and `makeOneway(`c,y`)`3. The first algorithm check if the **Constraint** c can be used to define **IntegerVariable** y and the second algorithm transforms c into a one-way constraint defining y. **(Need complexity arguments)**
The coefficients of the variables in constraint c are denoted $A(c)$ and the coefficients of variables in the objective function $f(\vec{y}) \in F$ denoted as $A(f(y))$. **(Maybe call it evalutation functions)** Then the coefficent of variable y in constriant c is $A(c, y)$.

Algorithm 2: Test if a constraint c can define a variable y

input : Constraint c and Variable y
output: Boolean

if c defines a oneway constraint **then**
  return *false*
end
if Number of integer variables not defined > 1 **then**
  return *false*
end
if `relation(`c`)` $==$ *Equal* **then**
  return *true*
end
foreach a in $A(f(y))$ **do**
  if $A(c, y) \cdot a > 0$ **then**
    return *false*
  end
end
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 y

input : Constraint c and Variable y
output: An Invariant

int $coef = A(c, y)$
$A(c) = A(c) \backslash \{A(c, y)\}$
$V(c) = V(c) \backslash \{y\}$
foreach $A(c, v)$ in $A(c)$ **do**
  $A(c, v) = A(c, v) \cdot \frac{-1}{coef}$
end
int $b = B(c)$
if `relation(`c`)` $==$ *Equal* **then**
  return `Sum(`$V(c)$,$A(c)$,b`)`
end
else
  Invariant inv =`Sum(`$V(c)$,$A(c)$,b`)`
  return `Max(`inv,b`)`
end

5.2 Dependency Digraph

6 Local Search Engine

6.1 Neighborhoods

6.1.1 Moves

6.2 Metaheuristics

7 Tests

8 Results

9 Conclusion