DM841
Discrete Optimization

Part 2 – Heuristics
Solvers

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Software Tools

- Modeling languages
  interpreted languages with a precise syntax and semantics
- Software libraries
  collections of subprograms used to develop software
- Software frameworks
  set of abstract classes and their interactions
  - *frozen spots* (remain unchanged in any instantiation of the framework)
  - *hot spots* (parts where programmers add their own code)

Software Tools

No well established software tools for Local Search:

- the apparent simplicity of Local Search induces to build applications from scratch.
- model and search are more interdependent than in CP and MILP: ie, constraints must be relaxed and this is hard to automatize
- the freedom of problem characteristics that can be tackled
- crucial roles played by delta/incremental updates which are highly problem dependent
- the development of Local Search is in part a craft, beside engineering and science. Very little if nothing has general validity
- However some attempts: Comet, LocalSolver, OscaR-CBLS

Software Tools

EasyLocal++	C++	Local Search
ParadisEO	C++	Local Search, Evolutionary Algorithm
OpenTS	Java	Tabu Search
Comet	Language	
LocalSolver	Modelling Language	
Google OR Tools	Libraries	
OscaR-CBLS	Modelling Language	

EasyLocal++	http://tabu.diegm.uniud.it/EasyLocal++/
ParadisEO	http://paradiseo.gforge.inria.fr
OpenTS	http://www.coin-or.org/Ots
Comet	http://dynadec.com/
LocalSolver	http://www.localsolver.com/
Google OR Tools	https://code.google.com/p/or-tools/
OscaR-CBLS	http://oscarlib.bitbucket.org/cbls.html

Outline

Comet was

Not Open Source

Developed by Pascal Van Hentenryck (Brown University), Laurent Michel (University of Connecticut), then owned by Dynadec.

It is not anymore in active development and available

Constraint-Based Local Search is

- Model
  - Incremental variables
  - Invariants (one-way constraints)
  - Differentiable objects
    - Functions
    - Constraints
    - Constraint Systems
- Search
  - Local Search
    - Iterative Improvement
    - Tabu Search
    - Simulated Annealing
    - Guided Local Search

OscaR has further developed the concept and architecture.

OscaR-CBLS

The only system really open [Björdal et al., 2015]

Based on Constraint-based local search by Van Hentenryck and Michel.

Constraint classification:

1. Implicit constraints: AllDifferent, GlobalCardinality with non-variable cardinalities, LinearEquality with unit coefficients, Circuit and Subcircuit.
2. One-way constraints defining invariants
3. Soft constraints

Dependency graph:
one-way constraints are topologically sorted based on the following digraph: each invariant is a node; there is an edge from a variable a to another variable b if a defines b via a one-way constraint

First general local search solver with a backend for MiniZinc.
An example for the N-queens problem:

```
val n = 8
val init = RandomPermutation(1..n)
var c = [Var(1..n,init.next()) | i in 1..n]
var cpi = [Invariant(c[i] + i) | i in 1..n]
var cmi = [Invariant(c[i] - i) | i in 1..n]
AllDifferent(cpi)
AllDifferent(cmi)
while(violation > 0){
    val i1 = selectOneOf(1..n)
    val i2 = selectOneOf(1..n)
    swapValues(c[i1],c[i2])
}
```

Neighborhoods

Neighborhoods are defined on independent variables only (roots of the dependency graph). Invariants are not handled by neighborhoods.
General purpose neighborhoods:

Binary variables:

- flip
- swap

Integer variables:

- one-exchange
- reassignment of a independent integer variable to another value in its domain

Constraint specific neighborhoods

- AllDifferent: swap between the values of two variables; reassignment of a variable to an unused value.
- GlobalCardinality: swap between the values of two variables; reassignment of a variable so that all cardinalities are satisfied
- Circuit: removal of one vertex from the circuit and insertion at some other point.
- Subcircuit: Circuit + removals without corresponding insertion; insertions of previously removed vertices
- LinearEquality: the value of one variable is decreased by some amount and the value of another variable is increased by the same amount

Search Procedure

- randomised initial assignment.
- neighbourhoods do not return all possible moves to the search procedure but are queried for a (random) best move
- Iterative improvement on general purpose neighborhoods: aims at minimising the global violation. Choose a variable and reassign to it the value that leads to the smallest global violation
- Tabu Search for satisfaction: objective function is neglected
- Tabu Search for optimization: ev. function: $w_1 \cdot v + w_2 \cdot f$, $w_1, w_2 \in \mathbb{Z}^+$.
  - initially $w_1 = w_2 = 1$
  - w_1 is is increased if the global violation is positive (i.e., there remain unsatisfied constraints) for a large number of iterations
  - w_2 is increased if the global violation is zero (i.e., all constraints are satisfied) but no better solution is found for a large number of iterations

Outline

Local Search Modelling Language

Enriched mathematical programming formulation:

- Boolean variables (0–1 programming)
- constraints (always satisfied) - decision between soft and hard left to user
- invariants
- objectives (lexicographics ordering)

Example (Bin-packing problem)

Input 3 items x, y, z of height 2,3,4 to pack into 2 piles A, B with B already containing an item of height 5.
Task Minimize height of largest pile

```
xA <- bool(); yA <- bool(); zA <- bool();
xB <- bool(); yB <- bool(); zB <- bool();
constraint booleansum(xA, xB) = 1;
constraint booleansum(yA, yB) = 1;
constraint booleansum(zA, zB) = 1;
heightA <- sum(2xA, 3yA, 4zA);
heightB <- sum(2xB, 3yB, 4zB, 5);
objective <- max(heightA, heightB);
minimize objective;
```

Black-Box Local Search Solver

- initial solution: randomized greedy algorithm (constraints satisfied)
- search strategy (standard descent, ie, iterative improvement + simulated annealing + random restart via multithreading)
- moves
  specialized for constraints and feasibility
- incremental evaluation machinery
  problem represented as a DAG: variables are roots, objectives leaves, operators induce inner nodes
  bredth-first search in DAG.

Local Solver

Example (Graph Coloring)

```
/* Declares the optimization model. */
function model(){
   x[1..n][1..k] <- bool();
   y[1..k] <- bool();

   // Assign color
   for[i in 1..n]
      constraint sum[l in 1..k](x[i][l]) == 1;

   for[c in 1..m][l in 1..k]
      constraint sum[i in 1..v[c][0]](x[v[c][i]][l
         ]) <= 1;

   y[l in 1..k] <- max[i in 1..n](x[i][l]);

   // Clique constraint
   obj <- sum[l in 1..k](y[l]);
   minimize obj;
}
```

Local Solver


```
/* Parameterizes the solver. */
function param(){
         if(lsTimeLimit == nil)
                lsTimeLimit=600;
  lsTimeBetweenDisplays = 10;
  lsNbThreads = 4;
  lsAnnealingLevel = 5;
}

/* Writes the solution in a file following the following format:
 * each line contains a vertex number and its subset (1 for S, 0
      for V-S) */
function output(){
         println("Write solution into file 'sol.txt'");
         solFile = openWrite("sol.txt");
         for [i in 1..n][l in 1..k]{
      if (getValue(x[i][l]) == true)
         println(solFile, i, " ", l);
   }
}
```

References

Benoist T., Estellon B., Gardi F., Megel R., and Nouioua K. (2011). **LocalSolver 1.x: a black-box local-search solver for 0-1 programming**. *4OR*, 9(3), pp. 299–316.

Björdal G., Monette J.N., Flener P., and Pearson J. (2015). **A constraint-based local search backend for minizinc**. *Constraints*, 20(3), pp. 325–345.

Van Hentenryck P. and Michel L. (2005). **Constraint-Based Local Search**. The MIT Press, Cambridge, USA.