Master informatique
Constraint programming (X8II060)

Université de Nantes
2012-2013

Tabu search for solving constraint satisfaction problems

Project – 3 TP sessions + personal work
Group 1: weeks 7, 10, 12 – Group 2: weeks 8, 11, 13

The purpose of this project is to implement a Tabu Search (TS) algorithm in `JaCoP` for solving constraint satisfaction problems. TS is a deterministic neighborhood search technique developped by Fred Glover in the 1970's. Given a search space $\mathcal{S}$, a fitness or cost function $f : \mathcal{S} \rightarrow \mathbb{R}$, and an initial *solution* $s \in \mathcal{S}$, TS explores the search space by succesive moves in a neighborhood of the current solution. The main components of TS are given below.

- A neighborhood structure determines the set of adjacent solutions that can be reached from the current solution s.
- A *tabu list* records forbidden moves (tabu moves). This list has a maximum size, which is an important parameter of TS.
- A tabu move leading to the best solution identified so far may be authorized (aspiration criterion).
- TS stops when an optimal solution is found or some conditions are met, for example, using a fixed number of iterations, a fixed amount of CPU time, a maximum number of successive moves such that the best known solution is not improved.

A basic tabu search algorithm is presented below. A candidate solution (line 6) is an element v in the neighborhood of the current solution s such that the move from s to v is not tabu or some aspiration condition holds. The best candidate solution is selected (line 8), which is the one having the minimum fitness. Then the current solution is updated (line 9) in every case. In particular, it is possible to move to a solution having a greater fitness, making it possible to espace local optima. The best known solution is possibly updated (line 11). Then the tabu list is updated, i.e., the current move is recorded and the oldest move is removed if the tabu list is full (line 13).

Algorithm 1 Basic TS algorithm

```
1:  $s \leftarrow$ generate a solution in $\mathcal{S}$                # initial solution
2:  $s^* \leftarrow s$                                                  # best known solution
3:  $k \leftarrow 0$                                                    # iteration number
4:  while stopping conditions not met do
5:      $k \leftarrow k + 1$
6:      $V \leftarrow$ subset of the neighborhood of $s$ such that tabu     # candidate solutions
7:              conditions are violated or aspiration conditions hold
8:      $\bar{v} \leftarrow \text{argmin}_{v \in V} f(v)$                # best candidate solution
9:      $s \leftarrow \bar{v}$                                           # update current solution
10:     if $f(\bar{v}) < f(s^*)$ then
11:         $s^* \leftarrow \bar{v}$                                     # update best solution
12:     end if
13:     update tabu list and aspiration conditions                     # update tabu conditions
14: end while
15: return $s^*$
```

In the CSP framework, a *solution* is a total assignment of the variables. The fitness function may be the number of constraints violated by the current assignment. A solution of the CSP is a total assignment with null fitness. We restrict our attention to CSPs with variables having integer domains.

Work

Form a team of 2 or 3 people. An archive containing Java sources and a report (pdf format, maximum 10 pages) must be uploaded to madoc no later than **Wednesday 3rd April 2013, 17h**.

Questions 1, 2, 3, 4 must be addressed. Question 5 should be addressed only if time allows.

1. Implement a first version of the algorithm such that:
    - the first solution is randomly generated;
    - a move changes the value of one variable;
    - the tabu list is then a sequence of assignments $x \to a$;
    - no aspiration condition is considered;
    - V contains all possible moves minus the tabu list;
    - TS is stopped if the fitness of the current solution is null, or a maximum number of iterations is reached, or a maximum number of steps without improving the best solution is reached.
2. Experiment this algorithm on known problems (in particular n-queens) and compare TS with complete search.
3. Consider aspiration conditions and make further experiments.
4. Plug the TS algorithm in a restarting procedure and make further experiments.
5. Propose an object-oriented design of TS algorithms and implement instances of the different algorithmic components.