INCOP 1.1 : User Manual

Bertrand Neveu

September 16, 2004

1 Problems and methods in the library

This software library (INCOP) solves combinatorial optimization problems with incomplete methods based on local search. All problems are mono-criterion minimization problems. This manual presents how to use the library in its version 1.1. Each implemented problem has its own running file, which is called by a command line. All the solving methods can be used in every executable. The parameters to enter for problems and methods are indicated below. For a general presentation of the library, read the presentation document.

A `makefile` file indicates how to build the running files under Linux using the source files.

1.1 Solved problems

INCOP can solve in its version 1.1 binary CSPs given in extension (random CSP, CSP in ds format), Nary CSP given in extension (wcsp format), the N-queens problem, graph coloring problem (CSP or impasse modeling), clique problem and CELAR frequency assignment problems (celar6, celar7, celar8, celar9, celar10 instances), car sequencing problems (as defined in CSP-LIB : http://www.4c.ucc.ie/ tw/csplib/) .

1.2 Solving Methods

Following local search methods, population based methods (gww : Go with the winners) , and hybrids methods (gww + local search) are implemented in the library.

For local search methods, the following metaheuristics are given in the library:

1. metropolis : a random walk with Metropolis acceptance criterion
2. simulated annealing : linear temperature descent
3. tabu list : a greedy walk with a tabu list

4. threshold accepting : worsening moves are accepted if evaluation delta is lower than a threshold that linearly goes down from threshold max to zero.

5. greedy : the moves improving or keeping constant the current configuration are accepted.

6. idw : Intensification-Diversification Walk : a part of the neighborhood is searched in order to find a move improving or keeping constant the current configuration value. The `end-neighborhood` parameter implements which deteriorating neighbor is chosen when no move is selected.

7. random walk : every move is accepted (can be interesting for the diversification walk in GWW algorithm).

The following solving methods are implemented :

1. gww (with threshold and random walk or local search) : the random walk is limited by the threshold. All implemented local search methods can be used (greedy, tabu list, metropolis ...) There exist 5 variants for managing the threshold descent.

   - gww : descent by a fixed rate until a minimum value (lower bound).
   - gww-killworst : descent by a fixed rate from the worst particle (which will be redistributed) until a minimum value (lower bound).
   - gww-adapt : descent which the K worst particles.
   - gww-mediandist : descent determined by a rate of the distance between the worst and the median particles.
   - gww-bestdist : descent determined by a rate of the distance between the worst and the best particles.

2. gww-nothreshold (gww without threshold managing, with random walk or local search)

2 Output file

The first parameter is the name of the **output file** where the results and the trace are written. If this parameter equals the string `arg`, the output file name is the concatenation of the arguments in the command line in a subdirectory of name `results` to be created before running the program.

3 Parameters for describing the problems

3.1 Graph Coloring

Graphs must be in the Dimacs standard format.
Executable : colorcsp or impassecolor (for impasse modeling)

Here is the parameter list :

- **file** : string : graph file name in Dimacs format
- **nb-colors** : integer : number of colors
  In case of use of the impasse modeling , 1 extra parameter is to be given:
  - **criterion** : boolean : 0 for the number of non assigned variables, 1 for the sum of degrees of non assigned variables.

3.2 Clique

The program searches for a clique of given size n in a graph. As for coloring problems, the graphs must be in Dimacs format.
Executable : maxclique

The neighborhood used is a swap type neighborhood.
Here is the parameter list

- **file** string : graph file name in Dimacs format
- **clique-size** :size of the clique to be found

3.3 Random CSPs

Random CSPs are generated by the urbcsp generator by Christian Bessière.
Executable : randomcsp

Here is the parameter list :

- **variable-number** : integer : number of variables
- **domain-size** : size of the domains
- **constraint-number** : number of constraints
- **tuple-number** : number of forbidden tuples in a constraint
- **generator-seed** : seed of the random number generator.
- **problem-number** : number of problems to be solved.

3.4 CELAR

The 5 CELAR problems that minimize a weighted sum of violated constraints (celar6, celar7, celar8, celar 9 and celar 10) are implemented.

Executable : celarincr (semi-incremental mode) or celarfullincr (full incremental mode)

- **file** : celar6.txt, celar7.txt , celar8.txt, celar9.txt, celar10.txt (file regrouping the data of each problem)

3.5 N-queens

The standard problem of placing N queens on a chess-board of size N*N, non attacking each other.
Executable : queen (most efficient implementation : swap move with non incremental evaluation)

- **queens-number** : integer : number of queens

3.6 Binary CSPs in extension

The problem is a binary CSP with constraints given in extension, solved as valued Max-CSP (minimizing a weighted sum of constraint violations). Format of the data file (ds by Simon de Givry).
Executable : wextcsp

- **file name** : string : file (in ds format) name

3.7 Nary CSP in extension

The problem is a nary CSP in extension solved as a weighted Max-CSP. (minimizing a weigted sum of constraint violations). Format of the data file (wcsp by Simon de Givry).
Executable : narycsp

- **filename** : name of the data file (in wcsp format)
- **lowerbound** : lower bound (stopping criterion when it is reached)

3.8 Car sequencing

Car sequencing problems as defined in the CSPLib. A configuration is a squence of vehicule classes. A move is an exhange in the sequence. Executables :

- carseq : basic version : each sequence in conflict for a given option has a cost 1.
- ficarseq : incremental version for computing the cost of a move : each sequence in conflict has a cost equal to the number of vehicles exceeding the option quota.
-
- **filename** : name of the data file (in format given by the CSPLib)

3.9 Latin square and spatially balanced latinsquare

Latin square problem (executable : latinsquare) and spatially balanced Latin square problem (executable : sblatinsquare) (as defined in the paper by Gomes et al in CPAIOR'04 proceedings)

- **size** : latin square size (integer)

4 Parameters of the solving methods

Two main types of methods are implemented : local search method dealing with one single configuration, and population based methods (gww). For population based gww method, one has to give a local search or random walk method for the particle walks.

- **seed** : integer : seed for the random number generator
- **trial-number** : integer : number of trials to perform on a problem (by incrementing the random seed at each trial)
- **method** : string : method used (local search: tabu, metropolis,simann,threshold, random, greedy, population based: gww gww-killworst,gww-adapt, gww-mediandist, gww-bestdist, gww-nothreshold)

  For local search methods, the parameters are the number of moves to perform, the metaheuristics parameter, and the neighborhood parameters).

- **move-number** : integer : number of moves to perform

  Depending on the metaheuristic used, the following parameters are to be entered.
  idw : descent (or idw) algorithm : no parameter to be entered (neighborhood paramaters (min-neighbors, max-neighbors, end-neighborhood) are used to implement variants of *idw* algorithm)

  metropolis : Metropolis acceptance condition

  - **temperature** : double : constant temperature used in the formula for accepting a move

  tabu

  - **tabu-length** : integer : length of the tabu list

  threshold : threshold accepting

  - **initial-threshold** : integer : maximum evalution-delta allowed at the beginning.

  simann : simulated annealing

  - **initial-temperature** : double : initial temperature (linear descent during the algorithm)

  Neighborhood Parameters

  - **var-conflict** : string : any variable (v) or a variable taking part of a conflict (cv)

- **min-conflict** : string : value minimizing the conflicts (mc) or random (v)
- **min-neighbors** : integer : minimum number of visited neighbors
- **max-neighbors** : integer : maximum number of visited neighbors
- **end-neighborhood** : integer : what to do when the neighborhood is exhausted (0 : nothing, 1 : first neighbor under the threshold, k : best neighbor among the k first neighbors under the threshold)
- **dynamic tuning** : integer : 0 : fix max-neighbors parameter, 1 : dynamic tuning of max-neighbors parameter

For population based methods, **gww or gww-killworst or gww-adapt or gww-mediandist or gww-bestdist or gww-nothreshold**, the following parameters are to be given :

- **particle-number** : integer : number of particles
- **grouped-test** boolean : walk only if the particle was regrouped before
- **last-move-descent** boolean : threshold descent at the last walk step
- **elitism** : boolean : putting the best particle in the population at each regrouping
- **walk-stop** : boolean : stopping the walk of a particle when no feasible move is found (no stagnation)

- Parameters for `gww` : regular threshold descent and
  for `gww-killworst` : descent with fixed rate from the worst particle
  * **threshold-descent-rate** double : (for example .01 when the thrshold is lowered by 1%)
  * **lower-bound** integer : gww : current-threshold = (1 - threshold-descent-rate) (current-threshold - lower-bound)
    gww-killworst : current-threshold = (1 - threshold-descent-rate) (worst-particle-cost - lower-bound)
- Parameters for `gww-adapt`
  * **killed-number** integer : number of particles at least killed at each regrouping (used to deternine the threshold descent)
- Parameter for `gww-mediandist`
  * **distance-median** double : distance rate from the worst particle to the median cost used to determine the threshold descent
- Parameter pour `gww-bestdist`
  * **distance-best** double : distance rate from the worst particle to the best cost used to determine the threshold descent
- Parameters for `gww-nothreshold`

    * **killed-number** integer : number of particles killed at each regrouping
    * **iteration-number** integer : number of iterations
- **walk-type** string : metaheuristics used for local search one among the implemented metaheuristics (see above) tabu,metropolis,random , greedy , simann , threshold
  Walk, metaheuristic and neighborhood parameters : see above

5 Trace Parameters

Trace-mode : 0 no trace, 1 trace at each best found and at each regrouping for `gww`methods.

6 Examples

Example of command lines :
colorcsp resqueen queen8_8.col 8 0 1 gww 100 0 1 1 0 0.05 200 metropolis 100 0.4 cv v 0 30 15 0 1
celarcsp rescelar8 celar8.txt 0 1 gww 100 0 1 1 0 0.05 320 metropolis 100 0.4 cv v 0 176 176 0 1
celarcsp rescelar8 celar8.txt 0 1 gww-adapt 100 0 1 1 0 5 metropolis100 0.4 cv v 0 176 2 0 1
celarcsp rescelar8 celar8.txt 0 1 metropolis 1000000 .4 cv mc 0 176 2 0 1
celarcsp arg celar8.txt 0 1 tabu 1000000 20 v v 20 176 2 0 0
randomcsp resrcsp 100 15 1000 52 0 10 0 4 metropolis 3000000 0.15 v v 0 800 1 0 1
randomcsp resrcsp 100 15 1000 52 0 10 0 4 idw 3000000 cv v 0 800 1 0 1