

Project sheet 4: Backjumping and Stochastic Local Search (30 points)

Prof. Dr. Bernhard Nebel, Dr. Stefan Wöfl,
Dr. Julien Hué, and Matthias Westphal

July 2, 2012

The code must be handed in by July, 16 in the git repository as usual.

1 Gaschnig's backjumping (10 points)

For this project, you will implement a simple version of gaschnig's backjumping (without Arc- or Path-Consistency). It is somehow simple to implement and the pseudocode for the implementation (as it appears in the Rina Dechter's book) is given here:

Algorithm 1: GASCHNIG'S BACKJUMPING

Input: A constraint Network $\mathcal{R} = \{X, D, C\}$

Output: Either a solution, or a decision that the network is inconsistent

```
1  $i \leftarrow 1$ 
2  $D'_i \leftarrow D_i$ 
3  $latest_i \leftarrow 0$ 
4 while  $1 \leq i \leq n$  do
5   instantiate  $x_i \leftarrow \text{SELECT-VALUE-GBJ}$ 
6   if  $x_i = \text{NULL}$  then
7      $i \leftarrow latest_i$ 
8   else
9      $i \leftarrow i + 1$ 
10     $D'_i \leftarrow D_i$ 
11     $latest_i \leftarrow 0$ 
12 if  $i = 0$  then
13   return inconsistent
14 else
15   return A total assignment
```

Gaschnig uses a marking technique to compute culprit. Each variable x_j maintains a pointer ($latest_j$) to the latest ancestor incompatible with any of its values. When

all the values from a domain have been tried, the algorithm backtracks to this $latest_i$ variable otherwise it behaves like a Backtrack algorithm.

The other change concerns the function where the value is selected: each time a value is likely to be given the GASCHNIG'S BACKJUMPING algorithm, its consistency with respect to the assignment is tested. If it is consistent then it is returned. If it is not consistent, the smallest prefix partial assignment that is contradictory with the value is computed and the $latest_i$ variable is changed accordingly (namely it is bigger than the previous $latest_i$).

We remind that \vec{a}_k denotes the partial assignment from x_1 to x_k .

Algorithm 2: SELECT-VALUE-GBJ

Input: $\mathcal{R} = \{X, D, C\}$: a constraint Network

Output: Either a solution, or a decision that the network is inconsistent

```

1 while  $D'_i \neq \emptyset$  do
2   select an arbitrary element  $a \in D'_i$  and remove  $a$  from  $D'_i$ 
3    $cons \leftarrow \text{true}$ 
4    $k \leftarrow 1$ 
5   while  $k < i$  and  $cons$  do
6     if  $k > latest_i$  then
7        $latest_i \leftarrow k$ 
8     if  $\text{not consistent}(\vec{a}_k, x_i = a)$  then
9        $cons \leftarrow \text{false}$ 
10    else
11       $k \leftarrow k + 1$ 
12  if  $cons$  then
13    return  $a$ 
14 return  $NULL$ 

```

1.1 Command Line

Your solver (providing the executable name is *./solver*) should run the backjumping procedure thanks to the command:

```
./solver --backjumping
```

Please note that this algorithm (under the present form) is not compatible with Consistency-Checking procedures. So please check that the command line does not allow both.

2 RandomWalk (10 points)

For this project you will implement a version of the SLS local search procedure using RandomWalk. The algorithm is given below. \vec{b} represents the best solution yet, it may be instantiated with a random total assignment run as a preprocess. The function $Score(C, \vec{a})$ which takes a set of constraints and a total assignment as parameters returns the number of violated constraints.

Algorithm 3: RANDOM-WALK

Input: $\mathcal{R} = \{X, D, C\}$: a constraint Network, MAX_TRIES and MAX_FLIPS: integers, p : a real number in $[0, 1]$

Output: Either a solution, or a decision that the network is inconsistent

```

1 for  $i \leftarrow 1$  to MAX_TRIES do
2    $\vec{a} \leftarrow$  a random assignment
3   if  $score(C, \vec{a}) < score(C, \vec{b})$  then
4      $\vec{b} \leftarrow \vec{a}$ 
5   for  $j \leftarrow 1$  to MAX_FLIPS do
6     if  $\vec{a}$  is a solution then
7       return  $\vec{a}$ 
8     else
9        $C \leftarrow$  a random violated constraint
10       $r \leftarrow$  a random real number  $[0, 1]$ 
11      if  $r < p$  then
12         $\langle x, a' \rangle \leftarrow$  a random pair variable-value where  $x \in scope(C)$ 
13        (and which is different from the current assignment)
14      else
15         $\langle x, a' \rangle \leftarrow$  a pair variable-value where  $x \in scope(C)$  and which
16        minimizes the number of new constraints break by the
17        assignment of  $x$  to  $a'$  (and which is different from the current
18        assignment)
19      Assign the value  $a'$  to  $x$ 
20 return  $\vec{b}$ 

```

Basically, the algorithm is allowed to perform a certain amount of tries (represented by MAX_TRIES). For each try, it starts by generating a random total assignment. If this assignment is a solution it is returned otherwise a change is made. If this change leads to a solution it is returned otherwise another is made (and so on... MAX_FLIPS times). This change is performed according to randomness factor p :

- with probability p : make a random change in the assignment
- with probability $1 - p$: make a greedy change in a variable that minimizes the number of newly violated constraints (constraints that were not violated in the

previous assignment and which will be violated if the new value is picked). Note that nothing is said about the reduction of the number of formerly violated constraints.

2.1 Command Line

Your solver (providing the executable name is *.solver*) should run the RandomWalk procedure thanks to the command:

```
./solver --randomwalk=X,Y,Z
```

where X will be interpreted as MAX_TRIES and Y will be interpreted as MAX_FLIPS and Z is an integer expressing p in percentage (i.e., $p = Z/100$).

Please note that this algorithm is not compatible with any other of the options (Consistency checking or backjumping or heuristics)

3 Evaluation

3.1 Backjumping (3 points)

Add into your repository the file *backjumping.txt* (dont forget *git add backjumping.txt*). This file must contain a table with the running times of the instances we provided with the max-cardinality heuristics. It should compare the running times of:

- the backtracking procedure with Maintaining Arc Consistency;
- Gaschnig's backjumping procedure;
- Simple backtracking procedure.

We suggest to put a time limit of 5 minutes to your tests thanks to the *ulimit* command.

After the table, write a short text (400-600 characters) commenting and comparing the results in the table.

3.2 RandomWalk (7 points)

Add into your repository the file *randomwalk.txt* (dont forget *git add randomwalk.txt*). You should perform a study of the parameters of the RandomWalk implementation you made based on the instances we provided in the newly created directory *randomwalk-instances*. The *randomwalk.txt* should aim at finding the best Z (or p) parameter for these implementation. You will thus need to run the provided instances (and provide the running times) for several possible value of X, Y and Z with the instances we provided. The study should address the following problems:

- Is the best value of Z depending on X, Y ? (Is this value moving when X or Y is different?)
- Is the best value of Z depending on the size of the instances?

Choose what is from your point of view the more appropriate value of p and explain your choice.