Catholic University of Louvain

Project 5

LINGI2365 - Constraint Programming

Auteurs :
Vanwelde Romain (3143-10-00)
Crochelet Martin (2236-10-00)

Groupe 7

Superviseurs :
Pr. Yves Deville
François Aubry

16 mai 2014

Table des matières

1 Explain the provided model (how are the constraints enforced, what do the global constraints used do, what do the decision variables represent, . . .)

First of all, the model creates three range variables :
- Customers from 1 to n.
- Depots from n+1 to n+K
- CustomersAndDepots from 1 to n+K

As we can see, we create one depot per vehicle (K vehicles) because we use the giant tour representation.

Decision variables

We can now take a look to decisions variables :

```
var<CP>{int} previous[CustomersAndDepots](cp,CustomersAndDepots);
```

Here, previous will contain for each customer/depot the id of the previous customer/depot.

```
var<CP>{int} routeOf[i in CustomersAndDepots](cp,1..K);
```

routeOf will contain for each customer/depot the id of the vehicle which will stop there.

```
var<CP>{int} service_start[i in CustomersAndDepots](cp,Horizon);
```

service_start will contain for each customer/depot the time when it will begin to be serviced.

```
var<CP>{int} departure[i in CustomersAndDepots](cp,Horizon);
```

departure will contain for each customer/depot the time when the vehicle will leave the current customer/depot.

```
var<CP>{int} totDist = minCircuit(previous,distance);
```

totDist will contain the minimum distance linking all customer/depot.

Constraints

```
forall(i in Customers) cp.post(routeOf[i] == routeOf[previous[i]]);
```

This constraint forces all the customers to be served by the same vehicle that the previous customer/depot. Thus, if we visualize it on the giant tour representation, it means that if we take a depot, this will have a specific vehicle, and that all the following customers (until the following depot) will have the same vehicle which will serve them.

```
forall(i in Depots) cp.post(routeOf[i] == i - Customers.getUp());
```

This constraint will assign to all depots a different vehicle.

```
forall(i in CustomersAndDepots){
        cp.post(service_start[i] == max(tw_start[i],departure[previous[i]] +
            distance[previous[i],i]));
        cp.post(service_start[i] <= tw_end[i]);
        cp.post(service_start[i] >= tw_start[i]);
}
```

This constraint express the fact that all customers/depot will begin to be served either at the beginning of the time window if the vehicle were on place sooner, or when the vehicle arrives (time when it leaves the previous customer plus the displacement time (which is equal to the distance)).
Then it expresses the fact that the service must begins before it ends (which is logical).
Last, it express the fact that for each customer/depot, the service will not be performed earlier than the beginning of its own starting window.

```
forall(i in Customers){
        cp.post(departure[i] ==  service_start[i] + service_duration[i]);
}
```

This constraint express the fact that the vehicle leaves a customer the duration of the service after having began the task.

```
forall(i in Depots){
        cp.post(departure[i] == 0);
}
```

This constraint express the fact that the vehicle will immediately leave the depot at time t0.

Global constraint

```
cp.post(multiknapsack(routeOf, demand, all(k in 1..K) Q));
```

2 Explain how you adapt the model to minimize the number of routes

A first adaptation could be create by maintaining the previous model, and adding a decision variable.

```
var<CP>{int} nOfRoutes(cp,1..K);
```

This variable will keep track the number of different routes (This is what we will try to minimize).

On our giant tour representation, a vehicle that is not used is represented by two depots following each other. The vehicle directly leaves the depot, and directly arrives to the other again.

```
cp.post(nOfRoutes == (sum(i in Depots) previous[i] <= Customers.getUp()) );
```

This constraint assign to nOfRoutes the number of depot which follows a customer. Thus, this represents the number of used vehicles that we want to minimize.

Since we want to first minimize the number of route, and the total distance, we minimized or problem with the following evaluation function.

```
minimize<cp> (nOfRoutes * upperBound + totDist)
```

where upperBound is the sum of all distances between all customers/customers and customers/depots. This way, we ensure that upperBound will always be bigger than totDist, and so a solution having a nOfRoute lower than an other will always be better considering this evaluation function.

After having created this program, we realized that we could optimize the program by using a dichotomic function. Instead of optimizing the number of vehicles and the number of routes at the same time, we decided to optimize first only the number of routes, and then the number of routes. The search on the number of vehicles is done with a dichotomic search, and the search on the route distance is done as an optimization problem.

The search on the number of vehicles :
We now give as arguments the upperbound for the tested values, and we add some additional constraints.

```
cp.post(nOfRoutes <= testedValue);
```

Is used to test our problem with an upper Bound.

```
forall(i in n+1..n+K-testedValue){
cp.post(previous[i+1]==i);
}
```

This constraint is used to gain over symmetry. We imposed that the vehicles that weren't used would be assigned as the first in the range. (And so those one have the previous var set to the previous depot in the id order).

When we throw the program, we first fix the lower bound to 9 vehicles. Indeed, the best number of vehicles actually founded on internet are most of time 10 (behave one which is 9) and so 9 is a good lower bound to begin the program, and to keep a chance to find a solution with better value that the actuals ones on internet (If a better solution is found, we always can throw the research again with a lower lower bound). The upper bound is set to 25.

The optimization of the routes length :
This optimisation is the same that the model that we received, with the constraints explained above that fixes the number of vehicles used.

The heuristic function :

```
forall(i in CustomersAndDepots) by (previous[i].getSize())
    tryall<cp>(v in CustomersAndDepots : previous[i].memberOf(v)) by (
        distance[i,v])
        label(previous[i],v);

————

forall(i in CustomersAndDepots) by (tw_end[i])
    tryall<cp>(v in CustomersAndDepots : previous[i].memberOf(v)) by (
        distance[i,v])
        label(previous[i],v);
```

heuristics_functions.co

For the first one, we try to first bind variables that have the least remaining values, and that by trying to assign them the closest depot/customer as previous value.

For the second one, we try to bind variables that has the sooner end window, and that by trying to assign them the closest depot/customer as previous value.

3 Explain each of your optimization procedures in detail (what, why and how)

4 Indicate and analyze your test results

Instance#	#Vehicles	Distance
C101.txt	10	82873
C102.txt	10	109019
C103.txt	10	129146
C104.txt	10	109296
C105.txt	11	103763
R101.txt	19	176622
R102.txt	19	168127
R103.txt	14	149756
R104.txt	12	130257
R105.txt	16	153319

Table 1 – Results of the first heuristic