LocalSolver : Modeling and solving difficult optimization problems by local search

LocalSolver is a math programming solver based on local search. LocalSolver can be used either through its APIs for classical programming languages (C++, Java, C#) or with a dedicated modeling language. In this exercise we will use the modeling language for solving several optimization problems.

This exercise consists in two parts. The first part will help you pick up the basic techniques on a simple problem. Then a non linear batch scheduling problem will be considered.

1 A simple knapsack problem

LocalSolver is launched from the command line and executes ".lsp" files. These files are made of four fonctions : **input**, **model**, **param**, **output**. The documentation of the modeling language is available at :
`http://www.localsolver.com/quicktouroflocalsolversmodeler.html`.

The function **input** allows declaring and reading the problem data. The file "kp_100_1.in" describes a knapsack instance in the following format :

- number of objects,
- weight of each object,
- value of each object,
- size of the knapsack.

The reading of this file is achieved in a few lines of code :

```
function input(){
        inFile = openRead("kp_100_1.txt");
        nbItems = readInt(inFile);
        for [i in 1..nbItems] weights[i] = readInt(inFile);
        for [i in 1..nbItems] values[i] = readInt(inFile);
        knapsackBound = readInt(inFile);
}
```

The function **model** allows declaring the model to be optimized. The knapsack problem reads as follows :

```
function model(){
        // 0-1 decisions
        for [i in 1..nbItems] x[i] <- bool();

        // weight constraint
        knapsackWeight <- sum[i in 1..nbItems](weights[i] * x[i]);
        constraint knapsackWeight <= knapsackBound;

        // maximize value
        knapsackValue <- sum[i in 1..nbItems](values[i] * x[i]);
        maximize knapsackValue;
}
```

The function **param** allows setting some search parameters. For the knapsack problem, we will just set a time limit of 5 seconds :

```
function param(){
        lsTimeLimit=5;
}
```

The function **output** allows printing some values at the end of the search. For instance we can print the value of the knapsack and the selected objetcs :

```
function output(){
        println("Knapsack value: " + getValue(knapsackValue));
        print("Knapsack items: ");
        for[i in 1..nbItems]{
                if(getValue(x[i]) == 1)
                        print(i + ", ");
        }
        println();
}
```

▷**Q1:** Write a "knapsack.lsp" model allowing to solve this knapsack problem. Test this model on the instance "kp_100_1.txt".

In order to solve this model, the launching command is `localsolver knapsack.lsp`. It starts a resolution by LocalSolver for 5 seconds and prints to the console :

```
Model:
  expressions = 479, operands = 602
  decisions = 100, constraints = 1, objectives = 1

Param:
  time limit = 5 sec, no iteration limit
  seed = 0, nb threads = 2, annealing level = 1

Objectives:
  Obj 0: maximize, no bound

Phases:
  Phase 0: time limit = 5 sec, no iteration limit, optimized objective = 0

Phase 0:

[0 sec, 0 itr]: obj = (0), mov = 0, inf < 0.1%, acc < 0.1%, imp = 0
[...]
[5 sec, 6127487 itr]: obj = (41700), mov = 12511689, inf = 42.6%, acc = 36.3%,
 imp = 207

6127487 iterations, 12511689 moves performed in 5 seconds
Feasible solution: obj = (41700)

Run output...
Knapsack value: 41700
Knapsack items: 1, 3, 4, 5, 6, [...] , 91, 94, 95, 96, 98, 99, 100,
```

▷**Q2:** Compare the number of iterations performed int 5 seconds to the number of decision variables on the different instances "kp_i_1.txt".

▷**Q3:** Add instructions to print (at the end of the resolution) the weight of objects in the knapsack and the capacity of the knapsack.

▷**Q4:** Add a secondary objective minimizing the number of items in the knapsack. By permuting objectives, show that they are considered in lexicographic order.

2 Batch scheduling

In this section we consider a batch scheduling problem with numerous industrial applications. We have a single production system (oven, cleaning machine,...). All demands are known and available at the beginning of the planning horizon. There are K orders to be satisfied. Each order is defined by a quantity r_k and a due date d_k. If order k is delivered after date d_k, a penalty β_k must be paid. No penalty is applied when the order is delivered earlier than its due date. The production is processed by batch whose size is between b et B. Each batch i has a starting date S_i and a completion date C_i. The duration of the batch $(C_i - S_i)$ is equal to its size (the resource produces one unit of product per unit of time). Each order produced in batch i is available at date C_i. The production cost of q_i^k units of of order k in batch i is $\beta_k q_i^k (C_i - d_k)$ if batch i is completed after the due date d_k, 0 otherwise. No batch setup cost is considered ($\pi = 0$). The problem consists in finding a schedule minimizing the sum of production costs.

2.1 Unsplittable orders

We will first consider than order cannot be splitted, that is to say that each order is assigned to a single batch.

▷**Q5:** Model (on paper) this problem as an optimization problem with binary decisions.

Instance files have the following format :
- number of orders K,
- minimum batch size b,
- maximum batch size B,
- quantity for each order (one value per order) r_k,
- due date (one value per order) d_k,
- tardiness penalty (one value per order) β_k.

▷**Q6:** Write a LocalSolver model reading instances for this batch scheduling problem and producing feasible solutions. To do this we will minimize a fake objective function , for instance : `minimize 1`. Test the model on instance file "instanceA.txt".

▷**Q7:** Introduce expressions defining from decision variables :start and end dates of each batch and tardiness penalties. Test this model on instances : "instanceA.txt", "instanceA2.txt", "instanceA3.txt".

2.2 Splittabled orders

Now we consider the case where each order can be splitted and assigned to one or several batchs. Instance file have the same format as before.

▷**Q8:** Model (on paper) this problem as an optimization problem with binary decisions. Compare the number of decisions in this model and in the previous one.

▷**Q9:** Create a new LocalSolver model to solve this problem and compare solutions found with the unsplittable model and the splittable model. Test the model on instance "instanceB.txt".

▷**Q10:** Test the model on instances "instanceC.txt" and "instanceD.txt". Comment the solution returned by LocalSolver.

When the problem is too constrained and finding an initial feasible solution is difficult, LocalSolver relaxes constraints and minimize the number of violated constraints. In order to guide the search for a feasible solution, we can also relax a constraint and add it as first optimization criterion. Such a constraint is called a soft constraint.

▷**Q11:** Relax the constraint on the minimum batch size and find a solution to instances "instanceC.txt" and "instanceD.txt".

Another approach consists in writing an extended formulation of the problem. For the batch scheduling problem we can generated all possible patterns for a batch. Then the problem consists

in assigning a pattern to each batch. The file "instanceD_etendue.txt" contains an extended formulation of "instanceD.txt" with the following format :

- number of orders K,
- minimum batch size b,
- maximum batch size B,
- quantity for each order (one value per order) r_k,
- due date (one value per order) d_k,
- tardiness penalty (one value per order) β_k.
- number of patterns M,
- quantity of order 1 in pattern 1, of order 2 in pattern 1, ... of order K in pattern 1
- quantity of order 1 in pattern 2, of order 2 in pattern 2, ... of order K in pattern 2
- ...
- quantity of order 1 in pattern M, of order 2 in pattern M, ... of order K in pattern M

▷**Q12:** Write an extended model for the batch scheduling problem and solve instance "instanceD_etendue.txt". Compare the number of decisions, expressions and the cost of solutions in the extended and splittable models.

▷**Q13:** Write a LocalSolver model reading a compact instance, generating the extended model and solving this extended model. Compare results obtained on instance "instanceE.txt" to results obtained with models with relaxed formulation.