OPC - Quelques Problèmes

1 Social Golfers

The problem is the task of scheduling m groups of n golfers over p weeks, such that no golfer plays in the same group as any other golfer twice (i.e. maximum socialisation is achieved).

```
class SocialGolfers extends Problem {
  int nWeeks = askInt("number of weeks");
  int nGroups = askInt("number of groups");
  int groupSize = askInt("size group = number of players per group");
  int nPlayers = nGroups * groupSize;

  void model() {
    // TO DO
  }
}
```

Etudier la possibilité d'écrire un modèle :

1. avec des variables 0/1
2. avec des variables entières (non 0/1)

2 Warehouse Location Problem

Le texte suivant, qui décrit le problème, correspond à la page CSPLib-prob034.

In the Warehouse Location problem (WLP), a company considers opening warehouses at some candidate locations in order to supply its existing stores. Each possible warehouse has the same maintenance cost, and a capacity designating the maximum number of stores that it can supply. Each store must be supplied by exactly one open warehouse. The supply cost to a store depends on the

warehouse. The objective is to determine which warehouses to open, and which of these warehouses should supply the various stores, suchthat the sum of the maintenance and supply costs is minimized.

As an example (from the OPL book), consider the following data :

```
fixed = 30;
Warehouses = { Bonn, Bordeaux, London, Paris, Rome };
nbStores = 10; //labeled from 0 to 9
capacity = [1,4,2,1,3]; // capacity is indexed by Warehouses

// supplyCost in indexed by Stores(0..9) and the set of Warehouses

supplyCost = [ [ 24, 11, 25, 30 ]
             , [ 28, 27, 82, 83, 74 ]
             , [ 74, 97, 71, 96, 70 ]
             , [ 2, 55, 73, 69, 61 ]
             , [ 46, 96, 59, 83, 4 ]
             , [ 42, 22, 29, 67, 59 ]
             , [ 1, 5, 73, 59, 56 ]
             , [ 10, 73, 13, 43, 96 ]
             , [ 93, 35, 63, 85, 46 ]
             , [ 47, 65, 55, 71, 95 ]
             ];
```

Then, an optimal solution has value 383, where :

— Stores of Bonn = 3

— Stores of Bordeaux = 8,6,5,1

— Stores of London = 9,7

— Stores of Paris =

— Stores of Rome = 4,2,0

Compléter le code suivant pour modéliser le problème. On pourra par exemple utiliser les contraintes `atMost`, `element` et `sum`.

```
class Warehouse extends Problem {
  String fileName = askString("Instance filename");

  int nWarehouses, nStores, fixedCost;
  int[] warehouseCapacities;
  int[][] supplyCosts;

  void data() {
    try (BufferedReader in = new BufferedReader(new FileReader(fileName))) {
      nWarehouses = Integer.parseInt(new StringTokenizer(in.readLine(), "NnbW=;").nextToken());
      nStores = Integer.parseInt(new StringTokenizer(in.readLine(), "NnbS=;").nextToken());
      fixedCost = Integer.parseInt(new StringTokenizer(in.readLine(), "fixed=;").nextToken());
      String line = in.readLine();
      if (line.startsWith("cap")) {
        warehouseCapacities = Kit.splitToInts(line, "capacity=[],; ");
        in.readLine();
      } else
        warehouseCapacities = Kit.arrayIntWith(nWarehouses, nStores);
      supplyCosts = new int[nStores][];
      for (int i = 0; i < supplyCosts.length; i++)
        supplyCosts[i] = Kit.splitToInts(in.readLine().trim(), "(\\s|\\[|\\]|,)+");
    } catch (IOException e) {
      Kit.exit("Erreur ouverture fichier " + e);
    }
  }

  void model() {
    data();
    // TO DO
  }
}
```

Des fichiers de données sont accessibles ici.

3 Conception de championnat sportif

Soit n un nombre pair d'équipes sportives devant se rencontrer au cours d'un championnat constitué de $2n-2$ journées (périodes). Le problème (single round-robin sport scheduling problem) consiste à indiquer pour une demi-saison constituée de $n-1$ journées la liste des rencontres opposant les n équipes lors de chaque journée en précisant pour chaque rencontre l'équipe qui joue à domicile et celle qui joue à l'extérieur. Évidemment, sur la demi-saison, chaque équipe rencontre exactement une seul fois chaque autre équipe. Par exemple, pour $n = 8$, une solution est :

Journée 1	Journée 2	Journée 3	Journée 4	Journée 5	Journée 6	Journée 7
1 - 2	2 - 4	1 - 4	2 - 8	1 - 6	2 - 5	1 - 8
4 - 7	3 - 1	3 - 5	4 - 6	3 - 2	4 - 3	3 - 6
6 - 5	5 - 8	6 - 2	5 - 1	5 - 7	6 - 8	5 - 4
8 - 3	7 - 6	8 - 7	7 - 3	8 - 4	7 - 1	7 - 2

Un break pour une équipe est définie comme deux matchs consécutifs joués à la maison ou comme deux matchs consécutifs joués à l'extérieur. Sur notre exemple ci-dessus pour $n = 8$, on peut entre autres observer un break pour l'équipe 3 aux journée 2 et 3. En ajoutant une fonction d'objectif consistant à minimiser le nombre de breaks sur l'ensemble des équipes et pour la demi-saison, on obtient une problème d'optimisation sous contraintes.

Proposer une modélisation et présenter les résultats obtenus pour résoudre les instances de ce problème (en satisfaction, puis en optimisation) pour différentes valeurs de n telles que e.g. $n = 6$, $n = 8$, $n = 10$, $n = 15$, $n = 20$, $n = 30$, $n = 50$, ...