Projet:
"Conception automatique de poster de photos"
Partie 1 : optimisation combinatoire

Résolution de Problèmes d'Optimisation
Master 1 I2L / WeDSci

Sébastien Verel
verel@univ-littoral.fr
http://www-lisic.univ-littoral.fr/~verel

Université du Littoral Côte d'Opale
Laboratoire LISIC
Equipe OSMOSE

Fonction objectif du problème mixte

Fonction objectif

Pour tout $X = (\sigma, x)$ avec $\sigma \in \mathcal{S}_n$ et $x \in \mathbb{R}^{+n}$,

$$f(X) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} s_{\sigma_i,\sigma_j}(x)\ d_{i,j}^2$$

$$\text{tel que } \sum_{k=1}^{n} x_k^2 = n$$

Complexité : $n(n-1)/2$

Partie combinatoire

Simplification

Dans cette partie, nous allons supposer que :

pour tout $k \in \{1, \ldots, n\}$, $x_k = 1$.

autrement dit, toutes les photos ont la même dimension.

Optimisation combinatoire

Alors le problème devient équivalent à un problème QAP :

Pour tout $\sigma \in \mathcal{S}_n$,

$$f(\sigma) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{i,j}^2 \; s_{\sigma_i, \sigma_j}$$

Complexité : $n(n-1)/2$

Classe générique des algorithmes d'optimisation

Créer un répertoire `algo` pour tous les algorithmes de recherche.
Créer une classe virtuelle `Search` générique.
Le critère d'arrêt commun est basé sur le temps.

```
class Search {
public:
  Search()  { }

  virtual void operator()(Solution & _solution) = 0 ;

  virtual void timeLimit(time_t limit) {
   timeLimit_ = limit;
  }

protected:
  time_t timeLimit_;
};
```

Recherche aléatoire

- Permet de mettre en place la génération aléatoire d'une permutation
- Algorithme de comparaison minimal, mais évidement pas efficace

Recherche aléatoire

```
#include <random>
#include <solution.h>
#include <posterEval.h>
#include <search.h>
#include <randomPermutation.h>
#include <uniformContinue.h>

class RandomSearch : public Search {
public:
  RandomSearch(std::default_random_engine & _rng, Eval & _eval):
    rng(_rng), eval(_eval) {  }

  virtual void operator()(Solution & _solution) { }

protected:
  std::default_random_engine & rng;
  Eval & eval;
};
```

Recherche aléatoire

```
virtual void operator()(Solution & _solution) {
  RandomPermutation random(rng, _solution.sigma.size());
  UniformContinue uniform(_solution.x.size());

  Solution s(_solution.sigma.size());
  uniform(s);

  while(time(NULL) < timeLimit_) {
    random(s);

    eval(s);

    if (s.fitness < _solution.fitness)
      _solution = s;
  }
}
```

Mise en place de opérateurs

- Créer un répertoire `operator`
- Créer la classe virtuelle générique `Operator`

```
#include <solution.h>

class Operator {
public:
    virtual void operator()(Solution & _solution) = 0;
};
```

Initialisation basique des variables continues

Rappel : la somme des x_i^2 doit être égale à n.

```
class UniformContinue : public Operator {
public:
    UniformContinue(unsigned int _n) : n(_n) {   }

    void operator()(Solution & solution) {
      solution.x.resize(n);

      for(unsigned i = 0; i < n; i++)
        solution.x[i] = 1.0;

    }

protected:
    unsigned int n;
};
```

Permutation aléatoire et tests

- Créer une classe `RandomPermutation` qui initialise une permutation aléatoirement.
- Créer un répertoire `exe` qui contient tous les exécutables.
- Créer un programme principal qui permet d'exécuter la recherche aléatoire. Le programme devra afficher la meilleure solution trouvée.
- Exécuter sur l'ensemble des instances la recherche aléatoire pendant 1 seconde 30 fois.
- Calculer la performance moyenne sur chaque instance.

Voisinage entre 2 permutations

Swap

$$\text{swap}_{u,v}(\sigma) = \sigma'$$

telle que :

$\sigma'_u = \sigma_v$ et $\sigma'_v = \sigma_u$

$\forall i \neq u$ et $i \neq v$, $\sigma'_i = \sigma_i$

$\sigma =$

2	3	1	5	4

$\text{swap}_{2,4}(\sigma) =$

2	5	1	3	4

Voisinage

$$\mathcal{V}(\sigma) = \{\text{swap}_{u,v}(\sigma) \mid \{u, v\} \in \{1, \ldots, n\}\}$$

$|\mathcal{V}(\sigma)| = n(n-1)/2$

Astuce : Evaluation incrémentale (dérivé première)

Variation de la fonction objectif par un swap

$$\Delta_{(u,v)} f(\sigma) = f(\text{swap}_{u,v}(\sigma)) - f(\sigma)$$

Après développement et simplification

$$\Delta_{(u,v)} f(\sigma) = \sum_{k=1, k\neq u, k\neq v}^{n} (d^2_{u,k} - d^2_{v,k})(s_{\sigma_v,\sigma_k} - s_{\sigma_u,\sigma_k})$$

avec $u < v$

$$\begin{aligned} \Delta_{(u,v)} f(\sigma) &= \sum_{k=1}^{u-1} (d^2_{u,k} - d^2_{v,k})(s_{\sigma_v,\sigma_k} - s_{\sigma_u,\sigma_k}) \\ &+ \sum_{k=u+1}^{v-1} (d^2_{k,u} - d^2_{v,k})(s_{\sigma_v,\sigma_k} - s_{\sigma_u,\sigma_k}) \\ &+ \sum_{k=v+1}^{n} (d^2_{k,u} - d^2_{k,v})(s_{\sigma_v,\sigma_k} - s_{\sigma_u,\sigma_k}) \end{aligned}$$

Complexité : 5 (n-2). Gain d'un facteur n

La mise a jour de la fonction objectif après un swap peut être calculée en complexité linéaire au lieu de quadratique.

En posant $\text{swap}_{u,v}(\sigma) = \sigma'$ avec $u < v$

$$\begin{aligned}\Delta_{(u,v)}f(\sigma) &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d_{i,j}^2 s_{\sigma'_i,\sigma'_j} - \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d_{i,j}^2 s_{\sigma_i,\sigma_j} \\ &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d_{i,j}^2 (s_{\sigma'_i,\sigma'_j} - s_{\sigma_i,\sigma_j})\end{aligned}$$

Lorsque $i \neq u$ et $j \neq v$, alors $s_{\sigma'_i,\sigma'_j} - s_{\sigma_i,\sigma_j} = 0$

$$\begin{aligned}\Delta_{(u,v)}f(\sigma) &= \sum_{j=u+1}^{n} d_{u,j}^2 (s_{\sigma'_u,\sigma'_j} - s_{\sigma_u,\sigma_j}) \\ &+ \sum_{j=v+1}^{n} d_{v,j}^2 (s_{\sigma'_v,\sigma'_j} - s_{\sigma_v,\sigma_j}) \\ &+ \sum_{i=1}^{u-1} d_{i,u}^2 (s_{\sigma'_i,\sigma'_u} - s_{\sigma_i,\sigma_u}) \\ &+ \sum_{i=1}^{v-1} d_{i,v}^2 (s_{\sigma'_i,\sigma'_v} - s_{\sigma_i,\sigma_v})\end{aligned}$$

or $\sigma'_u = \sigma_v$ et $\sigma'_v = \sigma_u$ et sinon $\sigma'_i = \sigma_i$

$$\begin{aligned}\Delta_{(u,v)}f(\sigma) &= \sum_{j=1}^{u-1} f_{u,j} (d_{\sigma_v,\sigma_j} - d_{\sigma_u,\sigma_j}) \\ &+ \sum_{j=1}^{v-1} f_{v,j} (d_{\sigma_u,\sigma_j} - d_{\sigma_v,\sigma_j}) \\ &+ \sum_{i=u+1}^{n} f_{i,u} (d_{\sigma_i,\sigma_v} - d_{\sigma_i,\sigma_u}) \\ &+ \sum_{i=v+1}^{n} f_{i,v} (d_{\sigma_i,\sigma_u} - d_{\sigma_i,\sigma_v})\end{aligned}$$

$$\begin{aligned}\Delta_{(u,v)}f(\sigma) &= 2(\sum_{j=1}^{u-1} f_{u,j}(d_{\sigma_v,\sigma_j} - d_{\sigma_u,\sigma_j}) \\ &+ \sum_{j=1}^{v-1} f_{v,j}(d_{\sigma_u,\sigma_j} - d_{\sigma_v,\sigma_j}) \\ &+ \sum_{i=u+1}^{n} f_{i,u}(d_{\sigma_i,\sigma_v} - d_{\sigma_i,\sigma_u}) \\ &+ \sum_{i=v+1}^{n} f_{i,v}(d_{\sigma_i,\sigma_u} - d_{\sigma_i,\sigma_v}))\end{aligned}$$

Ce qui donne :

$$\begin{aligned}\Delta_{(u,v)}f(\sigma) &= 2(\\ &\sum_{k=1}^{u-1} f_{u,k}(d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k}) + \sum_{k=u+1}^{n} f_{k,u}(d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k}) \\ - &\sum_{k=1}^{v-1} f_{v,k}(d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k}) - \sum_{k=v+1}^{n} f_{k,v}(d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k}))\end{aligned}$$

Lorsque $k = u$, alors par symétrie $d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k} = 0$

Lorsque $k = v$, alors de même $d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k} = 0$

d'où : $\sum_{k=u+1}^{n} f_{k,u}(d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k}) =$
$\sum_{k=u+1}^{v-1} f_{k,u}(d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k}) + \sum_{k=v+1}^{n} f_{k,u}(d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k})$

et $\sum_{k=1}^{v-1} f_{v,k}(d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k}) =$
$\sum_{k=1}^{u-1} f_{v,k}(d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k}) + \sum_{k=u+1}^{v-1} f_{v,k}(d_{\sigma_v,\sigma_k} - d_{\sigma_u,\sigma_k})$

D'où le résultat :

$$\begin{aligned}\Delta_{(u,v)}f(\sigma) &= 2(\sum_{k=1}^{u-1}(f_{u,k}-f_{v,k})(d_{\sigma_v,\sigma_k}-d_{\sigma_u,\sigma_k}) \\ &+ \sum_{k=u+1}^{v-1}(f_{k,u}-f_{v,k})(d_{\sigma_v,\sigma_k}-d_{\sigma_u,\sigma_k}) \\ &+ \sum_{k=v+1}^{n}(f_{k,u}-f_{k,v})(d_{\sigma_v,\sigma_k}-d_{\sigma_u,\sigma_k}))\end{aligned}$$