Projet "Conception automatique de poster de photos"

Master 1, 2021-2022
Version 1 du 02/02/2021

1 Description et but

1.1 But

Beaucoup d'outils d'aide à la conception d'album photo, ou de poster existent. Ces outils proposent principalement quelques patrons génériques et des outils d'amélioration de ces patrons: choix des couleurs, dimension des photos, dimension et position des emplacements des photos, etc. Seulement, ces applications ne proposent pas d'outils intelligents et automatiques de répartition des photos hormis quelques répartitions basiques comme l'ordre chronologique.

Il s'agit dans ce projet d'automatiser la phase de disposition des photos sur un poster (une affiche) et la taille des emplacements à l'aide de techniques d'optimisation. En définissant une fonction qui mesure le score de la disposition des photos, il s'agit d'utiliser des algorithmes d'optimisation efficaces pour trouver les meilleures solutions à ce problème.

1.2 Descriptif

A partir de la description des positions des photos sur un poster, et des informations sur une série de photos, il s'agit de trouver la meilleure disposition et taille des emplacements (voir Fig. 1).

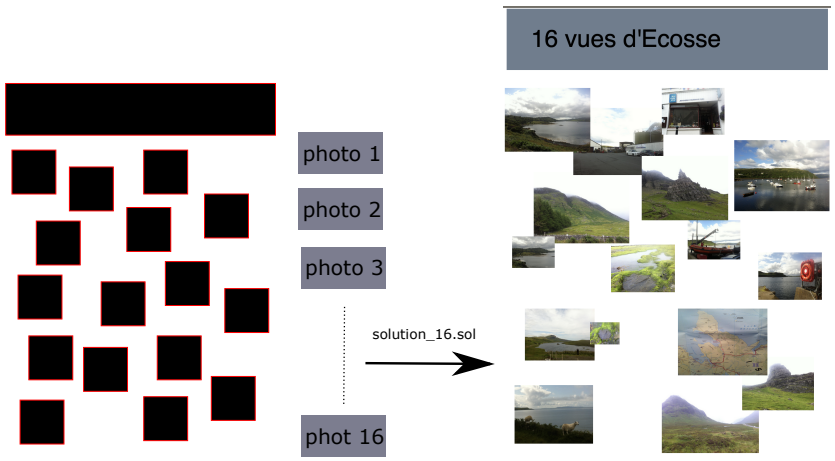

Figure 1: Principe de conception d'un poster de 16 photos.

Les fichiers au format json comme par exemple le fichier `poster_48_0.json` donne le descriptif des positions possibles et des photos (voir annexe pour les informations relatives aux photos). A partir d'un fichier de type "poster" et un fichier qui donne la disposition et la taille (fichier `solution_16.sol` dans la Fig. 1), le programme python `code/generator/buildPoster.py` permet d'obtenir un poster au

format svg (voir readme du git pour plus d'information). Les algorithmes d'optimisation ont pour but de créer les meilleurs fichiers de solutions.

1.3 Instances

Les posters au format json du répertoire `data`, `poster_n_0.json` avec $n \in \{6, 16, 32, 48, 55\}$ sont des posters créer "à la main" avec les n premières photos disponibles.

Des instances aléatoires ont été crées. Les paramètres de ces instances sont décrites dans le fichier `data/instances_param.csv`, et les fichiers correspondants sont dans le dossier `data/instances`. Le paramètre `density` correspond la proportion de surface occupée par les photos sur le poster. A priori, 0.4 devrait donner des instances plus facile à résoudre que 0.6.

Vous pourrez tester vos algorithmes sur l'ensemble de ces instances.

2 Définition du problème d'optimisation

2.1 Contexte

Le problème proposé est un problème à variable mixte. Les problèmes d'optimisation à variable mixte sont des problèmes d'optimisation où une partie des variables sont des variables continues appartenant à $\mathbb{R}$, et une parties des variables sont discrètes (variables ordinales, catégorielles, ou d'un espace combinatoire). Ces problèmes se rencontrent dans de nombreux problèmes de conception où par exemple certaines variables sont des longueurs et certains variables sont des types de matériaux. Les algorithmes d'optimisation pour ce genre de problèmes ne sont pas encore très développés, et beaucoup de principes algorithmiques sont encore à inventer.

2.2 Espace de recherche et fonction objectif

Une solution au problème est donné par une permutation σ donnant la disposition des n photos sur les emplacements, et un vecteur de nombre réel x de dimension n donnant le facteur d'agrandissement de l'emplacement.

L'espace de recherche de ce problème est donc $\Omega = \mathcal{S}_n \times \mathbb{R}^{+n}$, c'est-à-dire que les n premières variables représentent une permutation de $\{1, \ldots, n\}$, et les n dernières variables sont des nombres réels positifs. La fonction objectif proposée à minimiser est définie par :

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

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

$$\text{tel que } \sum_{u=1}^{n} x_u^2 = n$$

$d_{i,j}$ représente la distance entre les photos i et j, et $s_{u,v}(x)$ est le score entre les positions u et v. Plus la distance est petite, plus les photos sont "proches". Plus le score est grand, plus les positions mutuelles sont "pertinentes". Ainsi, la minimisation de la fonction objectif a but pour but associer les grands scores avec les petites distances.

La distance $d_{i,j}$ dépend des deux couleurs principales coul1 et coul2 de la photos, et des mots clés associés à chaque photo (20 pour chaque photo) :

$$d^2_{i,j} = \alpha_1\ d^2_{\text{coul1}}(i,j) + \alpha_2\ d^2_{\text{coul2}}(i,j) + \alpha_3\ d^2_{\text{tag}}(i,j)$$

où $d^2_{\text{coul}} = \frac{1}{255^2}((r_i - r_j)^2 + (g_i - g_j)^2 + (b_i - b_j)^2)$ avec (r, g, b) le code RGB de la couleur, et $d^2_{\text{tag}}(i,j)$ est le carré de la proportion de mots clés différents entre les deux photos. Les coefficients α_1, α_2 et α_3 pondèrent l'importance de chacune des distances. Ici, $\alpha_1 = 1/3.5$, $\alpha_2 = 0.5/3.5$, et $\alpha_3 = 2/3.5$.

La fonction score $s_{u,v}(x)$ dépend de la distance des centres des emplacements, de la taille des emplacements u et v, de leur équilibre de tailles et de l'intersection des emplacements. Les variables x_u s'interprète comme le facteur d'agrandissement de l'emplacement u. Ainsi un emplacement original de largeur w_u et de hauteur h_u possède sur le poster une largeur de $x_u w_u$ et une hauteur de $x_u h_u$. Ainsi, la contrainte $\sum_{u=1}^{n} x_u^2 = n$ exprime que la surface totale des photos est constante, égale à la surface originale des emplacements du poster. Le score est définie de la manière suivante :

$$s_{u,v}(x) = \frac{a_u(x_u) + a_v(x_v) + \beta_1 E a_{u,v}(x_u, x_v) - \beta_2 a_{u\cap v}(x_u, x_v)}{D_{u,v}^2}$$

$D_{u,v}$ est la distance euclidienne entre les centres des emplacements u et v. $a_u(x_u)$ et $a_v(x_v)$ sont les aires des emplacements u et v : $a_u(x_u) = x_u^2 w_u h_u$, et $a_v(x_v) = x_v^2 w_v h_v$. $E a_{u,v}(x_u, x_v)$ est la moyenne géométrique des aires a_u et a_v : $E a_{u,v}(x_u, x_v) = \sqrt{a_u(x_u) a_v(x_v)} = x_u x_v \sqrt{a_u(1) a_v(1)}$, et enfin $a_{u\cap v}(x_u, x_v)$ est l'aire de l'intersection entre les emplacements u et v.

Le score s'interprète de la manière suivante. Il est inversement proportionnel au carré de la distance des centres des emplacements. Il varie donc comme une force de gravité entre les centres. Le score augmente avec la somme des aires des emplacements. Plus les aires sont grandes, mieux seront vues les photos et le score est d'autant plus grand. Le terme de moyenne géométrique permet d'équilibrer les tailles des emplacements. En effet, pour une surface égale de deux emplacements, la moyenne géométrique est maximale lorsque les emplacements ont la même surface. Enfin, nous voulons éviter la superposition des photos et une pénalité est comptée proportionnellement à la surface de l'intersection. Les coefficients β_1 et β_2 pondèrent l'importance des termes. Ici par défaut, $\beta_1 = 1$ et $\beta_2 = 2$.

2.3 Code

Le code fourni sur gitlab à l'adresse `https://gitlab.com/verel/projet-optimisation-2022` contient la description d'une solution dans la classe `Solution`, et la classe `PosterEval` qui capable de lire une instance au format json et de calculer la valeur de la fonction d'évaluation (fitness) correspondante.

Le programme `main` donne un exemple d'utilisation minimale. Il est possible de modifier la classe `Solution` si besoin. Il est aussi possible de proposer d'autres fonctions d'évaluation qui tiennent compte d'autres caractéristiques des photos ou des emplacements.

3 Travail à réaliser

Le projet est à faire en binôme ou en monôme. Le code, les scripts et le rapport sont à rendre à l'intermédiaire d'un git (privé ou public). Le rapport doit être au format pdf en indiquant vos noms dans le document.
Vous répondrez aux questions de manière argumentée et synthétique. Pour mémoire, la plagiat est interdit, facile à détecter et sera sanctionné.

3.1 Première partie : optimisation combinatoire

Dans cette partie, les variables continues sont fixées à la valeur $x_u = 1$ pour tout emplacement u. Le problème est alors équivalent à un problème d'assignement quadratique (QAP) [1].

Consignes : A rendre le xx/xx/2022.

Voisinage : Le voisinage de deux permutations peut être défini par un opérateur swap qui consiste à échanger 2 éléments de la permutation. Formellement, l'opérateur swap est défini par : $\text{swap}_{u,v}(\sigma) = \sigma'$

telle que : $\sigma_u^{'} = \sigma_v$ et $\sigma_v^{'} = \sigma_u$ et $\forall i \neq u$ et $i \neq v$, $\sigma_i^{'} = \sigma_i$.

Par exemple, si $\sigma =$ | 2 | 3 | 1 | 5 | 4 | alors $\text{swap}_{2,4}(\sigma) =$ | 2 | 5 | 1 | 3 | 4 |

Le voisinage est alors défini par $\mathcal{V}(\sigma) = \{\text{swap}_{u,v}(\sigma) \mid \{u, v\} \in \{1, \ldots, n\}\}$. La taille du voisinage est donc de $n(n-1)/2$.

Questions :

1.a - Écrire en pseudo-code et développer la recherche locale Hill-climbing Best-Improvement où l'opérateur de voisinage est défini par l'échange de 2 éléments (swap).

1.b - Écrire en pseudo-code et développer la recherche locale ILS ou une recherche Tabou basée sur le Hill-Climbing précédant. La perturbation pourra être donnée par l'échange de k éléments aléatoire.

1.c - Évaluer les performances de votre algorithme sur les instances fournies.

1.d - En suivant les équations (1) et (2) de la section 3.1 de l'article [1], développer l'évaluation incrémentale du calcul de la fitness des solutions voisines.

3.2 Deuxième partie : optimisation continue

Consignes : A rendre le xx/xx/2022.

On fixe la permutation σ à l'identité, c'est à dire $\sigma(i) = i$ pour tout i.

Questions : Implémentez un $(1 + 1)$-ES avec adaptation du pas selon la règle des 1/5 résolvant ce problème sous contraintes. Vous stopperez votre algorithme lorsque le progrès sur f a été inférieur à 10^{-6} au cours des 200 dernières itérations, et observerez la variation de ce que trouve votre algorithme en le faisant démarrer sur différents points.

Pour gérer les contraintes, utilisez la méthode de réparation suivante :

- pour tout $i \in [1..n]$, si $x_i < 0$ alors $x_i = -x_i$,
- ensuite si $\sum_{i=1}^{n} x_i^2 = 0$ alors $x_i = 1$,
- enfin si $\sum_{i=1}^{n} x_i^2 > 0$ alors $x_i = x_i \sqrt{\frac{n}{\sum_{i=1}^{n} x_i^2}}$.

3.3 Troisième partie : optimisation à variable mixte

Dans cette partie, il s'agit de développer un algorithme d'optimisation qui manipule à la fois les variables discrètes et les variables continues de l'espace de recherche Ω.

Questions :

3.a - Écrire en pseudo-code et développer un algorithme d'optimisation qui alterne une phase d'optimisation combinatoire avec une phase d'optimisation continue.

3.b - Écrire en pseudo-code et développer un algorithme d'optimisation qui utilise un voisinage qui combine le changement de variables discrètes et continues.

3.c - Évaluer et comparer les algorithmes que vous avez développés.

4 Annexe

Les informations sur chacune des photos sont contenus dans le fichier au format json comme `poster_48_0.json`.

Les informations données sont :

- index : index (commençant à 0) utilisé pour identifier la photo lors de la construction de l'album,
- name : nom du fichier contenant la photo dans le dossier `html/img`,
- id : index défini par l'appareil photo. Il donne aussi l'ordre chronologique des prises de vue,
- size: dimensions en pixel de la photo originale,
- date: date de la prise vue (année, mois, jour, heure, minute et seconde),
- color1 : en réduisant la palette des couleurs à 16 couleurs, couleur la plus fréquente de la photo,
- color2 : en réduisant la palette des couleurs à 16 couleurs, deuxième couleur la plus fréquente de la photo,
- greyavg : niveau de gris moyen de la photo lorsqu'elle est convertie en noir et blanc,
- note: note entre 0 et 100 attribuée par des personnes (0 est la note la plus basse et 100 la plus haute),
- ahash, phash et dhash : empreintes (hash) calculées sur les photos. Les empreintes sont des chaines binaires de 64 bits écrites en hexadécimal. A partir des caractéristiques de la photo, une fonction de hachage calcule l'empreinte. Lorsque les empreintes sont identiques, les photos sont proches voire identiques elles aussi. 3 types de hachage sont calculées[1] : average, perceptive[2][3], et difference hash[4].
- ahashdistance, phashdistance et dhashdistance : distances de Hamming (normalisées entre 0 et 1) entre les photos basées sur les empreintes précédentes qui mesurent de différentes manières la similarité des images. Lorsqu'une distance est nulle, les photos sont censées être identiques modulo quelques transformations ou détails. La liste des distances aux autres photos est donnée dans l'ordre des index.
- tags : mots tag associés à l'image. En utilisant les outils de la société clarifai[5], les tags sont calculés automatiquement à partir d'algorithmes d'apprentissage de type réseau de neurones profond. Le champs `classes` donne la liste des mots tag et le champ `probs` donne la liste des probabilités que chaque mot soit associé à l'image.

References

[1] Éric Taillard. Robust taboo search for the quadratic assignment problem. *Parallel computing*, 17(4-5):443–455, 1991.

[1] Le code python utilisé est celui de Jens Segers `https://github.com/jenssegers/imagehash`

[2] Zauner, Christoph: Implementation and Benchmarking of Perceptual Image Hash Functions. Master's thesis, Upper Austria University of Applied Sciences, Hagenberg Campus, 2010.

[3] Voir aussi le billet du blog du Dr. Neal Krawetz `http://www.hackerfactor.com/blog/?/archives/432-Looks-Like-It.html`

[4] Issu également d'un billet du blog du Dr. Neal Krawetz `http://www.hackerfactor.com/blog/index.php?/archives/529-Kind-of-Like-That.html`

[5] `http://www.clarifai.com/`