Appariement de squelettes pour la reconnaissance d'objets

Introduction

Le squelette est un modèle permettant de représenter un objet 2D ou 3D. Il s'agit d'une structure fine, centrée à l'intérieur de l'objet. Son avantage, notamment en 2D, est qu'il représente de façon simple la topologie de la forme (*cf.* figure 1). Ainsi, il est aisé à partir d'un squelette de compter le nombre de « membres » d'un objet.

Dans ce projet, cette propriété est exploitée pour effectuer de l'appariement d'objets. Pour ce faire, il est nécessaire de traduire dans un premier temps la notion de squelette en termes de graphes. Pour ce faire, chaque branche du squelette est représentée par une arête du graphe, et chaque point de jonction par un sommet du graphe (*cf.* figure 1). Les squelettes considérés dans ce sujet n'ont pas de cycle, chaque graphe est donc un arbre.

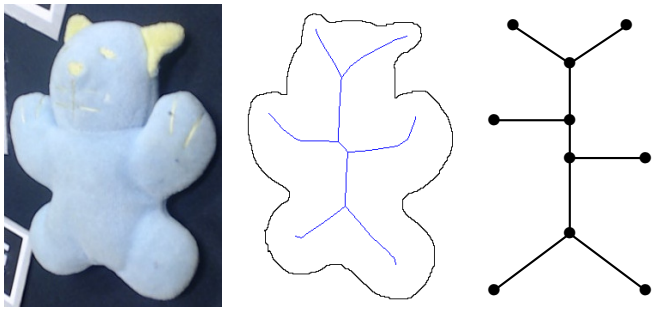

Figure 1 – Squelette d'un objet et graphe associé.

Le but de ce projet sera d'estimer la distance d'édition entre deux graphes G_1 et G_2, représentant deux squelettes. La distance d'édition est le nombre minimum de transformations nécessaires (contractions ou insertions d'arêtes, substitutions de nœuds) pour obtenir le graphe G_2 à partir du graphe G_1 (*cf.* figure 2).

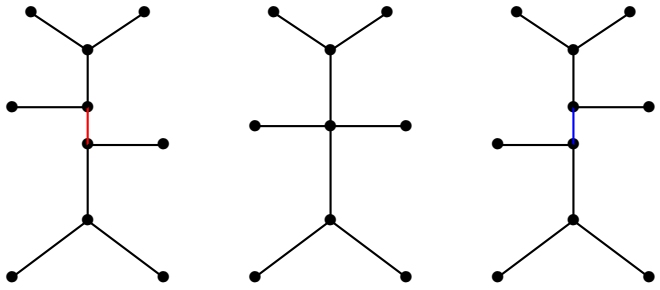

Figure 2 – Passage du graphe de gauche au graphe de droite à l'autre en deux opérations : contraction de l'arête rouge puis insertion de l'arête bleue.

Pour estimer cette distance d'édition, il est nécessaire dans un premier temps de mettre en correspondance un premier sommet sur G_1 et G_2. De plus, pour simplifier la mise en correspondance, on ordonne les différentes feuilles du graphe en sens direct (*i.e.* par angle croissant) autour du sommet d'origine.

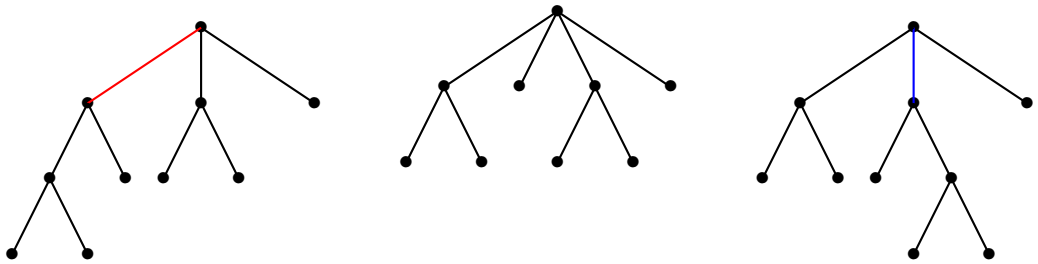

Figure 3 – Illustration du passage de l'arbre de gauche à l'arbre de droite après mise en correspondance d'un premier sommet. Les feuilles sont ordonnées en sens direct autour du sommet mis en correspondance.

La section 1 est consacrée à l'implémentation des différentes opérations sur les graphes (contraction, insertion d'une arête, substitution de nœuds). Par la suite, en section 2, une première version de l'algorithme est décrite, ne comprenant que la mise en correspondance des arbres, sans les modifier. La version avec modification de l'arbre (*i.e.* avec contraction/insertion d'arêtes) est l'objet de la section 3. Enfin, une méthode d'accélération de l'algorithme est demandée en section 4. Les prototypes de chaque fonction sont donnés dans le fichier projet.mli.

1 Opérations sur les graphes

Avant de commencer l'implémentation de l'algorithme de mise en correspondance, il est nécessaire de coder les différentes opérations applicables sur un arbre. Ces opérations sont la contraction/insertion d'une arête, mais aussi l'appariement de sommets dans les deux graphes. On fera en sorte que chaque opération soit réversible.

1.1 Appariement de nœuds

Pour apparier des nœuds dans les graphes, on utilise le module Mark. On travaillera sur des arbres, tels que le label de chaque nœud soit un entier non nul. Ainsi, pour associer le nœud v_1 de label 1 dans le premier graphe, et le nœud v_3 de label 3 dans le second graphe, on attribue la marque 3 à v_1, et la marque 1 à v_3.

Question 1:
Programmez une fonction ASSOCIATE, prenant en paramètre deux sommets, et leur attribuant une marque à chacun.

Pour faire en sorte que cette opération soit réversible, il faut pouvoir retirer la marque attribuée à chaque sommet. Cela consiste à attribuer à chaque sommet la marque 0. Comme le label de chaque

sommet est non nul, un sommet ayant la marque 0 est considéré comme non attribué.

Question 2:
Programmez une fonction SEPARATE, prenant en paramètre deux sommets, et leur retirant leur marques.

1.2 Contraction/Insertion d'arête

Les opération de contraction et d'insertion d'arête sont des opérations réciproques. Chaque arête possède un sommet o et un sommet a. L'opération de contraction d'une arête consiste en trois points :

- Retrait de l'arête (o, a)
- Pour tout voisin s de a (autre que o), retrait de l'arête (a, s) et création de l'arête (o, s)
- Retrait du sommet a.

Afin de pouvoir être réversible, il est nécessaire de donner en sortie de la fonction l'ensemble des voisins S_a de a (autres que o), avant l'opération.

Question 3:
Programmez une fonction CONTRACT, respectant les différentes conditions énoncées ci-dessus.

Enfin, l'opération d'insertion d'une arête permet de restaurer le graphe à l'état précédant la contraction, en n'oubliant pas de ré-attribuer à a ses successeurs.

Question 4:
Programmez une fonction INSERT, prenant en paramètre un graphe, deux sommets o et a, l'ensemble S_a, et inversant l'opération de contraction. La succession des opérations de contraction et d'insertion doit donner le même graphe.

2 Test d'égalité de deux arbres

Avant d'élaborer un algorithme évaluant la distance d'édition entre deux arbres, il est nécessaire de comprendre comment faire un test d'égalité entre deux arbres, ayant un point apparié. L'algorithme 1 présente une méthode récursive pour effectuer ce test.

La fonction ORDERED_SUCC donne les successeurs d'un sommet donné, ordonnés en sens direct autour de ce sommet. Cette hypothèse permet de restreindre la recherche des opérations pour passer d'un graphe à l'autre : en effet, on associe implicitement le sous arbre gauche du premier arbre, au sous arbre gauche du second arbre, ce qui réduit drastiquement le nombre d'associations possibles.

Pour finir, cette fonction renvoie un couple comprenant un booléen, ainsi que la liste des nœuds mis en correspondance.

Algorithm 1: Fonction auxiliaire pour le test de l'égalité entre deux arbres.

```
1  Function EQUALS_AUX(G1, v1, G2, v2)
2      S1 = UNMARKED(ORDERED_SUCC(G1,v1))
3      S2 = UNMARKED(ORDERED_SUCC(G2,v2))
4      if EMPTY(S1) and EMPTY(S2) then
5          return (true,[ ])
6      else if EMPTY(S1) or EMPTY(S2) then
7          return (false,[ ])
8      else
9          h1 = HEAD(S1)
10         h2 = HEAD(S2)
11         ASSOCIATE(h1,h2)
12         (bh,lh) = EQUALS_AUX(G1,h1,G2,h2)
13         (bq,lq) = EQUALS_AUX(G1,v1,G2,v2)
14         SEPARATE(h1,h2)
15         lf = CONCAT(lh,lq)
16         return (bh and bq,(h1, h2) :: lf)
17     end
18 end
```

Algorithm 2: Test de l'égalité entre deux arbres.

Data: G_1, G_2 : Deux arbres
v_1, v_2 : Sommets mis en correspondance dans les deux arbres
Result: b : Booléen vrai si les arbres sont identiques
l : Liste des associations de nœuds

```
1 ASSOCIATE(v1,v2)
2 (b,lr) = EQUALS_AUX(G1,v1,G2,v2)
3 l = (v1, v2) :: lr
4 SEPARATE(v1,v2)
```

Question 5:
Implémentez la fonction EQUALS_AUX décrite par l'algorithme 1, et la fonction equals, décrite par l'algorithme 2. On utilisera pour ceci la fonction ORDERED_SUCC fournie, pour obtenir les sommets suivants en ordre direct autour du nœud considéré. Vous coderez aussi la fonction UNMARKED, qui à partir d'une liste de sommets renvoie les sommets non marqués.

3 Évaluation de la distance d'édition

Afin d'évaluer la distance d'édition entre deux graphes, il faut déterminer le nombre de contractions et d'insertions d'arêtes minimum nécessaire pour passer d'un graphe à l'autre. On peut toutefois reformuler le problème, sur la base de la figure 3 : sur cette figure, on constate que une contraction de l'arête rouge est nécessaire pour passer de l'arbre de gauche à l'arbre du milieu, et que une contraction de l'arête bleue est nécessaire pour passer de l'arbre de droite à l'arbre du milieu.

Ainsi, il ne s'agira non plus de déterminer le nombre de contractions et d'insertions nécessaires pour passer d'un arbre à l'autre, mais de déterminer le nombre de contractions nécessaires sur chaque arbre

pour que ces arbres deviennent identiques.

Dans la fonction EQUALS_AUX, on cherche à chaque étape à marquer un sommet. Pour l'estimation de la distance d'édition, on cherchera à chaque étape à tester toutes les opérations possibles, qui sont au nombre de trois :

— marquage des sommets h_1 et h_2
— contraction de l'arête de gauche (v_1, h_1) dans le premier graphe
— contraction de l'arête de gauche (v_2, h_2) dans le second graphe

Question 6:
En vous inspirant de la fonction EQUALS_AUX et EQUALS, programmez les fonctions DISTANCE_AUX et DISTANCE, qui évaluent la distance d'édition entre deux graphes. On n'oubliera pas, après chaque contraction d'arête, de restaurer le graphe à son état initial. Cette fonction renverra un quadruplet de données (c, l_0, l_1, l_2), où c désigne le nombre d'éditions nécessaires pour passer d'un graphe à l'autre, l_0 désigne la liste des sommets appariés entre les deux graphes, l_1 désigne la liste des arêtes contractées dans le premier graphe, et l_2 la liste des arêtes contractées dans le second graphe.

4 Accélération de l'algorithme

On peut remarquer que l'algorithme implémenté en section 3 peut être un peu lent. Ceci est dû au fait que toutes les possibilités de contractions d'arêtes sont explorées. Cependant, l'exploration de toutes les possibilités est inutile ici. En effet, si on a réussi à passer d'un arbre à l'autre en c éditions, on sait que la distance d'édition est au plus c, et que toutes les éditions demandant plus de c contractions ne sont pas pertinentes (*cf.* figure 4).

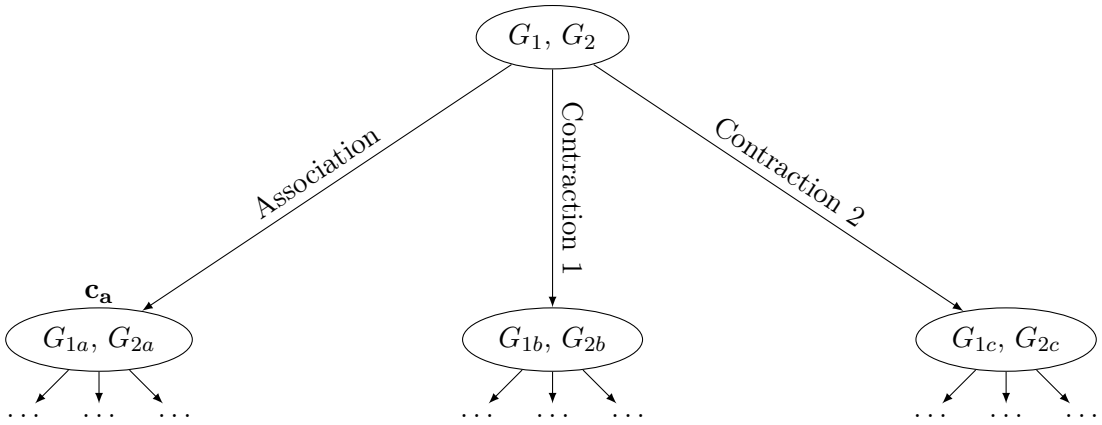

FIGURE 4 – Arbre des transformations possibles. Après avoir estimé le score $\mathbf{c_a}$ associé à la branche de gauche, il est inutile d'explorer les possibilités demandant plus d'opérations que le score indiqué.

Question 7:
Vous rendrez un rapport de 5 pages maximum, où vous transcrirez :
- Un bilan du travail accompli, montrant votre compréhension des sections 1 à 3 (environ 1 page)
- Une présentation de la problématique de la section 4, et de votre algorithme permettant de solutionner ce problème (environ 2 pages)
- Un ensemble de tests comparant les performances entre cet algorithme et l'algorithme de la section 3 (environ 1 page)
- Une conclusion synthétisant vos travaux sur ce projet (environ 1 page)

Question 8:
Programmez une fonction DISTANCE_OPTI qui restreint l'arbre des éditions possibles.

5 Evaluation du projet

Le projet sera réalisé en binôme.

Vous devez rendre un code commenté, auquel vous ajouterez les tests de vos fonctions.

Echéances :
- 28/02 au soir : premier rendu sur moodle (sections 1 à 3)
- 05/03 : séance de tests, rendu du code sur moodle
- 10/03 au soir : rendu du rapport et du code pour la section 4

Barême (approximatif) :
- Code des sections 1 à 3 : 10pts
- Évaluation de la séance de tests : 5pts
- Rapport et code de la section 4 : 5pts