Recherche d'un chemin Hamiltonien de coût minimal approché

Abel Douzal

9 juin 2021

1 Introduction et motivation du problème

Le problème de recherche d'un chemin Hamiltonien de coût minimal joue un rôle crucial pour réduire la consommation d'énergie dans de nombreuses applications. Par exemple, pour optimiser la distribution de produits ou de services à domicile, le traitement par drones de zones agricoles ou le ramassage de déchets marins. En revanche, la complexité exponentielle du problème de recherche d'un chemin Hamiltonien de coût minimal constitue un frein à son utilisation en pratique. Ainsi, l'objectif de ce projet est d'étudier des approches de complexités polynômiales approchant en coût la solution exacte et de coût en temps raisonnable. Pour cela, je propose plusieurs algorithmes approchant la solution exacte d'un chemin Hamiltonien dont les performances et complexités sont comparées au travers de jeux de données simulées et réelles de dimensions variables. La suite de ce document est structurée comme suit : d'abord je discute les limitations du problème en Section 2. En Section 3, j'introduis quelques approches de parcours élémentaires et de référence, puis je présente la méthode proposée d'approximation de la solution exacte et l'algorithme associé. Dans la Section 3.3, je discute des éléments d'implémentation de la méthode en OCaml. Les résultats des expérimentations menées sont présentés et discutés en Section 4. Enfin, je conclus ce travail en Section 5.

2 Limitations du problème (NP complétude)

Un problème d'optimisation $\mathbb{P}$ peut être défini par un triplet $\mathbb{P} = (I, R, f)$, où I est l'ensemble des instances du problème, R est l'ensemble de ses résultats et $f : I \times R \to \mathbb{R}$, la fonction à **minimiser**. On appelle solution du problème une fonction $s : I \to R$ associant un résultat à une instance du problème. On notera S l'ensemble de ces solutions.

Une solution $s \in S$ est dite optimale ssi : $\forall i \in I, \forall g \in S, \quad f(i, s(i)) \leqslant f(i, g(i))$

On suppose dans la suite que le problème est bien défini, c'est-à-dire qu'il existe toujours un résultat optimal à toute instance du problème. Dans toute la suite, on confondra les notions d'algorithme et de solution.

La recherche d'un chemin hamiltonien de coût minimal, partant d'une source donnée, est un problème NP-complet [1] se rapportant à celui bien connu du voyageur de commerce. Il existe une

solution optimale à ce problème qui consiste en l'exploration exhaustive de tous les chemins avec mémoïsation [4]. Cette solution est en coût temporel très élevé en $O(n^2 2^n)$ où n est le nombre de sommets du graphe.

Par ailleurs, étant donné le problème $\mathbb{P} = (I, R, f)$ de solution optimale Opt, on appelle ρ-approximation une solution $Algo$ de coût temporel polynômial qui renvoie des solutions au pire cas à un facteur ρ de l'optimal Opt, à savoir :

$$\forall i \in I \quad f(i, Opt(i)) \leqslant f(i, Algo(i)) \leqslant \rho f(i, Opt(i))$$

On appelle facteur de ρ-approximation d'une solution $Algo \in S$:

$$\rho_{Algo} = \sup_{i \in I} \left| \frac{f(i, Algo(i))}{f(i, Opt(i))} \right|$$

Théorème : (Admis) Dans le cas du problème du chemin Hamiltonien de coût minimal, en admettant $P \neq NP$ et $\rho \geqslant 1$, il n'existe pas de ρ-approximation de la solution optimale du voyageur de commerce.

3 Méthode proposée

Pour la recherche d'un chemin Hamiltonien de coût minimal approché, on modélise les n points-cibles à parcourir par les sommets d'un graphe pondéré et non orienté G, sans cycles de poids négatifs. On note $M = [m(i,j)]_{i,j} \in \mathscr{M}_{nn}(\mathbb{R})$ la matrice d'adjacence du graphe G de terme général $m(i,j) \in [0,1]$ correspondant au poids de l'arête reliant les sommets i et j. Les poids des arêtes peuvent correspondre à des distances ou des dissimilarités normalisées [2] entre les points-cibles ou à une toute autre mesure quantifiant le coût de parcours entre les points cibles.

Étant donné que le problème de la ρ-approximation est difficile, on propose dans ce travail une méthode de coût temporel polynômial dont l'objectif est d'offrir des résultats -à coût temporel modéré- meilleurs, sur différentes distributions de points cibles, que les méthodes de référence telles que celle utilisant l'algorithme de Kruskal ou un parcours par minimisation locale naïve.

3.1 Quelques méthodes élémentaires et de référence d'approximation d'un chemin Hamiltonien de coût minimal

Tout d'abord, j'explore trois méthodes de parcours élémentaires fondées sur l'optimisation pas-à-pas du coût du chemin. Dans la première, j'étudie deux types de parcours : P_{kk} et P_{k1} où le chemin est construit en choisissant les k sommets suivants optimaux et en les parcourant ; puis on teste les chemins de longueur k à partir du sommet courant et on avance de, respectivement, k et 1 pas. Dans la seconde $\mathrm{P}_{Kruskal}$, il s'agit d'utiliser l'algorithme Kruskal [5] pour obtenir le parcours de l'arbre couvrant de poids minimal. Un chemin pré-Hamiltonien, potentiellement moins bon qu'un chemin Hamiltonien, est ainsi obtenu. Dans la troisième approche P_{alea} on adopte un parcours aléatoire consistant à choisir, à chaque étape du chemin, le sommet suivant selon une loi uniforme.

3.2 Méthode proposée : Approche Récursive par Partitionnement (ARP)

Pour la recherche d'un chemin approximant un chemin Hamiltonien de coût minimal, je propose une Approche Récursive par Partitionnement (ARP). Le principe de cette approche consiste à partitioner le graphe initial G en K sous-graphes $\{G_k\}_{k=1}^{K}$ en utilisant l'algorithme de clustering PAM [6, 3]. Étant donné la matrice d'adjacence M, l'algorithme PAM (ou k-medoids) permet de partitionner l'ensemble des sommets V en K classes $\{V_1, ..., V_K\}$ par optimisation du problème suivant :

$$\min_{V_1,...,V_K} \sum_{k=1}^{K} \sum_{s \in V_k} d(s, r_k)$$

où, r_k est le sommet de référence de V_k :

$$r_k = \arg\min_{r \in V_k} \sum_{s \in V_k} d(s, r)$$

Dans le cas d'une matrice d'adjacence représentant les distances entre points cibles, chaque sous-graphe obtenu est composé des sommets les plus proches, et les sommets distants sont affectés à des sous-graphes différents. Le meilleur parcours est alors obtenu par une recherche récursive dans chaque sous-graphe G_k puis entre les sous-graphes. Je détaille dans ce qui suis les principales étapes de l'algorithme ARP.

Étant donné un Graphe G de n sommets et un nombre de sous-graphes $K = \lfloor \sqrt{n} \rfloor$, on applique la méthode PAM à la matrice d'adjacence M du graphe G pour l'obtention de K classes, chaque classe représentant l'ensemble des sommets d'un sous-graphe. L'algorithme ARP est appliqué récursivement à chacun des sous-graphe G_K obtenu. Étant donné que les sous-graphes sont indépendants, cette étape peut être appliquée séquentiellement à chacun des sous-graphes ou être parallélilsée. On note $\{L_1, ..., L_K\}$ les meilleurs parcours obtenus dans les sous-graphes. Pour déterminer, le meilleur parcours entre les K sous-graphes, où chaque sous-graphe G_k est représenté par les sommets (s_i, s_j) délimitant le chemin L_k de coût $c(L_k)$, on définit $MG = [mg(k, k')]_{k,k'} \in \mathscr{M}_{KK}(\mathbb{R})$ la matrice d'adjacence d'un graphe de sommets G_k et dont le poids de l'arête $mg(k, k')$ entre les sommets G_k et $G_{k'}$ est défini :

$$mg(k, k') = \frac{c(L_k) + c(L_{k'})}{2} + \max\left(m(deb_{L_k}, deb_{L_{k'}}), m(deb_{L_k}, fin_{L_{k'}}), m(fin_{L_k}, deb_{L_{k'}}), m(fin_{L_k}, fin_{L_{k'}})\right)$$

deb_L et fin_L donnant respectivement le premier et le dernier élément de la liste L. Le meilleur parcours entre les sous-graphes est alors obtenu par application de l'algorithme ARP au graphe MG. La dernière étape de l'algorithme permet de concatener les parcours intra et inter sous-graphes obtenus en retournant, dans certains cas, la liste L_k par l'utilisation de la fonction miroir. L'algorithme 1 résume les principales étapes de l'approche ARP proposée.

3.3 Éléments d'implémentation en OCaml

Les graphes sont représentés sous la forme de matrices d'adjacence ou de listes de voisins. On utilisera de manière préférentielle la matrice d'adjacence. Le type des différentes fonction implémentées sont :

```
type grapheM=float array array;;
type grapheV=(int*float) list array;;
type choix=grapheM -> bool array -> int -> int list;;
```

Les différents algorithmes de parcours sont :

```
choix_k_1 : choix
choix_k_k : choix
parcours : grapheM -> choix -> int list

aleatoire : grapheM -> int list
kruskal : grapheM -> (int * int) list
acpm :  grapheV -> int list
```

L'algorithme PAM est composé de deux étapes séquentielles BUILD et SWAP. Dans l'étape BUILD, un premier partitionnement est obtenu, il servira de configuration initiale à l'étape SWAP. Ces étapes sont totalement implémentées sous OCaml.

```
pam : grapheM -> int -> int array
voronoi : grapheM -> int list -> int list array
```

On précise que ici :

- *acpm* désigne la méthode proposée ARP
- *kruskal* désigne $\mathrm{P}_{Kruskal}$
- *aleatoire* désigne la méthode P_{alea}
- Il est fixé une variable globale $k = 3$ pour les algorithmes $choix_{k1}$ et $choix_{kk}$ qui désignent respectivement les méthodes P_{31} et P_{33}
- L'algorithme PAM renvoie les centres (médoïdes) et *voronoi* la partition associée au diagramme de voronoï de ces mêmes centres.

4 Expérimentation et résultats

Pour l'évaluation de la méthodes proposée, j'ai considéré sept jeux de données publiques, un jeu de données réelles et 6 jeux de données simulés, chacun représentant une distribution différente des points cibles. Le premier jeu de données $USArrests$ [1] donne la dispersion de 50 états des états unis (les points cibles) selon leur profils de criminalités (Figure 1). Les cinq jeux de données suivants $S1$ (140), $D31$ (160), $Jain$ (373), $Flame$ (240) et $Agglomeration$ (788) sont simulés [2], le sixième jeu $Agglopetit$ (140) est une version plus petite du jeu Agglomeration. Le nombre des points cibles composant chaque jeu est indiqué entre parenthèses. La Figure 2, visualise la répartition des points-cibles des cinq jeux de données considérés.

4.1 Comparaison des approches étudiées

La méthode proposée ARP est comparée aux méthodes élémentaires et alternatives : $\mathrm{P}_{Kruskal}$, P_{alea} et P_{kk} pour $k = 3$ notées, respectivement, P_{31} et P_{33}. Dans le tableau 1, est indiqué le coût du

1. https ://forge.scilab.org/index.php/p/rdataset/source/tree/master/csv/datasets/USArrests.csv
2. http ://cs.joensuu.fi/sipu/datasets/

chemin obtenu pour chaque jeu de données et pour chaque méthode. Pour chaque jeu de données, le meilleur coût obtenu est indiqué en gras et le symbole "-" indique une interruption des calculs non terminés au bout de 4h. Enfin, pour chaque méthode, est indiqué en dernière ligne le rang moyen obtenu au travers de l'ensemble des jeux de données.

4.2 Analyse des résultats

A partir des résultats de ces expériences, nous pouvons voir que la méthode ARP obtient le meilleur coût pour 4 jeux sur 7 et un second meilleur score, proche du meilleur score, pour les 3 jeux restants. ARP obtient, par ailleurs, le meilleur rang en moyenne (1.85) suivi dans l'ordre par P_{31}, P_{33}, $\mathrm{P}_{Kruskal}$ et P_{alea}. Les méthodes P_{31} et P_{33} obtiennent les meilleurs coûts pour 3 jeux de données, en revanche, elles ont un coût d'exécution bien supérieur sur de grandes données. Par exemple, 5000 fois supérieur dans le cas de *Jain* ; pour le jeu *Agglomeration*, le temps mis par ARP est de 33s, par $\mathrm{P}_{Kruskal}$ et P_{alea} d'environ 3s contre des calculs non terminés et qui ont dû être interrompus au bout de 4h pour P_{33} et P_{31}. $\mathrm{P}_{Kruskal}$ réalise l'avant dernier score et les plus mauvais scores sont obtenus par P_{alea}, comme attendu.

Du point de vue de la complexité temporelle, deux groupes d'algorithmes se distinguent, dans le premier on trouve P_{kk} et P_{k1} qui ont une complexité temporelle selon le nombre de points en entrée n et de k au moins en $O\left(\frac{n^{k+1}}{k}\right)$ ce qui donne $O(n^4)$ pour P_{33} et P_{31}. Le second groupe composé de *Kruskal* qui est en $O(n\log(n))$, P_{alea} en $O(n^2)$, et ARP en $O(n^3\log_2(\log_2(n)))$.

5 Conclusion

L'agorithme ARP, bien que ne donnant pas une solution optimale au problème du chemin Hamiltonien de coût minimal, il donne à coût temporel modéré, un chemin hamiltonien de coût relativement faible dans un graphe pondéré. Cet algorithme peut, par ailleurs, être optimisé en adoptant d'autres conditions d'arrêt (pour un petit nombre de sommets) que celle implémentée.

Références

[1] Olivier Carton. *Langages formels, calculabilité et complexité*, volume 28. Vuibert, 2008.

[2] Michel Marie Deza and Elena Deza. Encyclopedia of distances. In *Encyclopedia of distances*, pages 1–583. Springer, 2009.

[3] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, chapter 10.4.3, 2012.

[4] Christoph Dürr and Jill-Jênn Vie. *Programmation efficace*. ellipses, 2016.

[5] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1) :48–50, 1956.

[6] Erich Schubert and Peter J Rousseeuw. Faster k-medoids clustering : improving the pam, clara, and clarans algorithms. In *International Conference on Similarity Search and Applications*, pages 171–187. Springer, 2019.

6 Annexe : Algorithme, Figures et Tableaux

Algorithm 1 ARP (M)

n = taille(M)

{Condition d'arrêt }

if $n \leqslant 3$ **then**

    choisir le meilleur chemin parmi les au plus trois cas possibles.

end if

$\boldsymbol{K} = \lfloor \sqrt{\boldsymbol{n}} \rfloor$

{Partitionnement de G en K sous-graphes }

(G_1,...,G_K) $\leftarrow$ PAM (M, K)

{Recherche des parcours L_k à l'intérieur des sous-graphes}

for $k = 1, ..., K$ **do**

    $M_k = Adjacence(G_k)$

    L_k = ARP(M_k)

end for

{Generation du graphe connectant les sous-graphes}

$$\forall\ k, k' \in \{1, ..., K\} \quad mg(k, k') = \frac{c(L_k)+c(L_{k'})}{2} + \max \begin{cases} m(deb_{L_k}, deb_{L_{k'}}) & (1) \\ m(deb_{L_k}, fin_{L_{k'}}) & (2) \\ m(fin_{L_k}, deb_{L_{k'}}) & (3) \\ m(fin_{L_k}, finL_{k'}) & (4) \end{cases}$$

$MG = [mg(k, k')]_{k,k'} \in \mathscr{M}_{KK}(\mathbb{R})$

{Recherche du parcours R entre les sous-graphes}

$R \leftarrow$ ARP(MG)

{Construction du parcours global P par concaténation des parcours L_k et R}

for $k = 1, ..., K-1$ **do**

    if $mg(R[k], R[k+1])$ est issu des cas (1) ou (2) **then**

        $P_{R[k]}$ = Miroir($L_{R[k]}$)

    else

        $P_{R[k]}$ =$\mathrm{L}_{R[k]}$

    end if

end for

if $mg(R[K-1], R[K])$ est issu des cas (2) ou (4) **then**

    $P_{R[K]}$ = Miroir($L_{R[K]}$)

else

    $P_{R[K]}$ =$\mathrm{L}_{R[K]}$

end if

return ($P_{R[1]} \uplus \cdots \uplus P_{R[K]}$) *{ $\uplus$ opérateur de concaténation}*

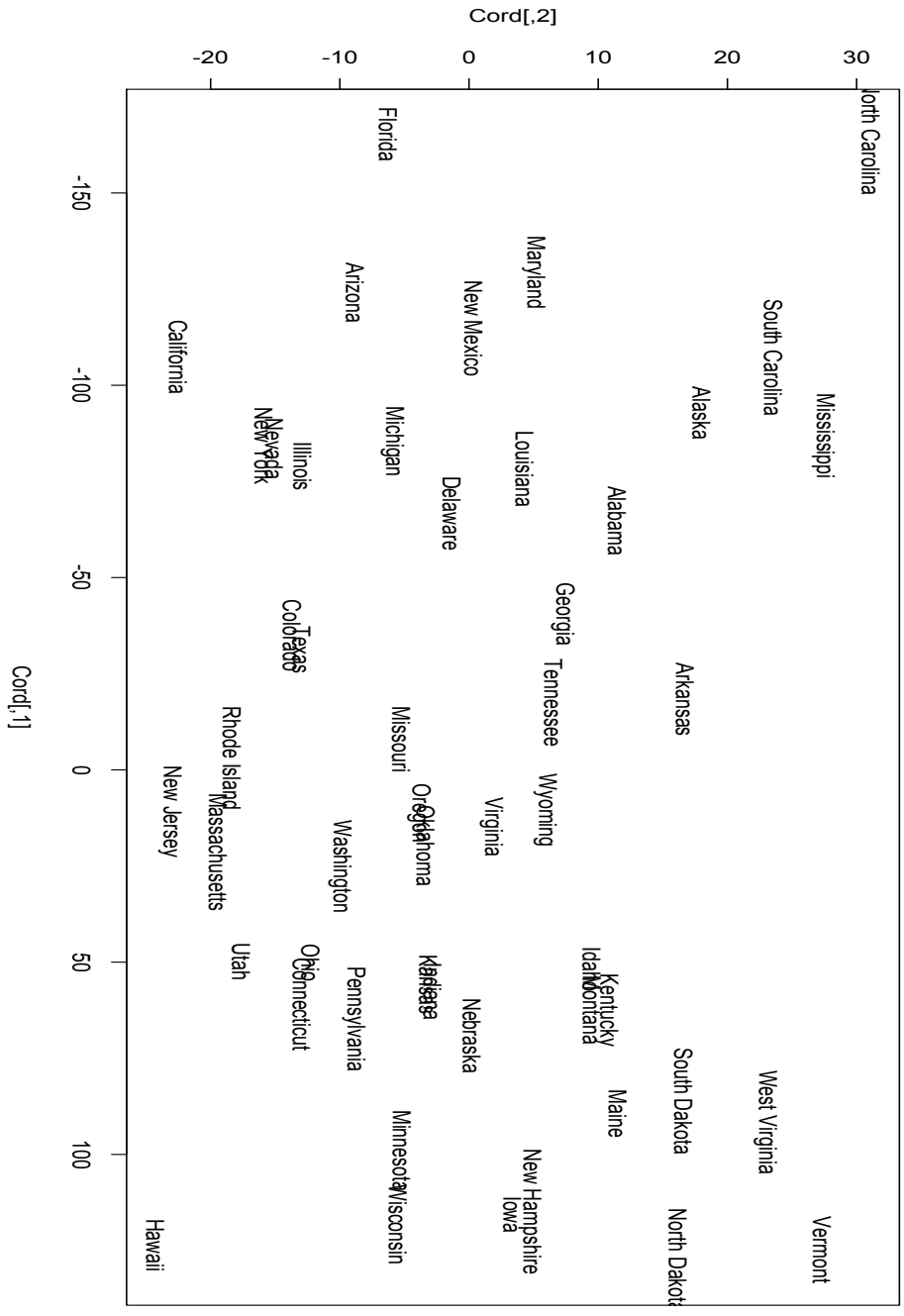

FIGURE 1 – Distribution des données USArrests

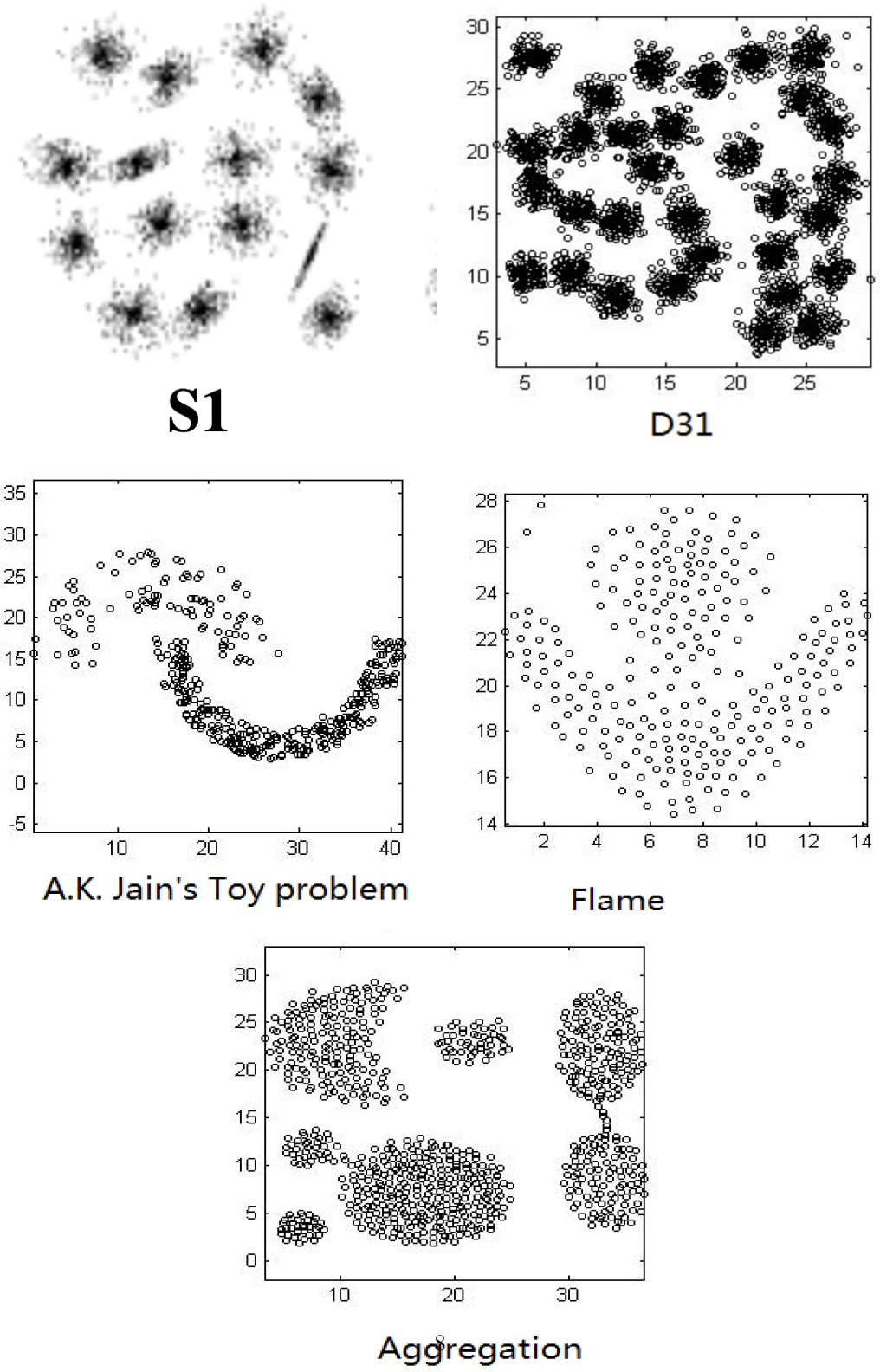

FIGURE 2 – Distributions des jeux de données simulées

TABLE 1 – Coût des chemins

	ARP	KRUSKAL	P_{33}	P_{31}	P_{alea}
USARREST	**1172.690**	1548.785	1395.760	1232.828	5017.268
S1	**737.235**	892.175	760.789	839.537	6050.899
D31	**132.408**	176.618	163.079	152.859	1864.839
JAIN	472.571	496.364	**429.15**	435.932	5494.012
FLAME	259.989	298.018	217.298	**213.889**	1451.059
AGGLOMERATION	**935.419**	1007.35	-	-	12688.146
AGGLOM-PETIT	308.199	386.038	302.609	**296.209**	2518.529
Rang. Moy	1.85	3.85	2.42	2.14	5