Résolution du problème de Sac à dos 0-1

Rapport de projet

Kévin Barreau

28 octobre 2015

Résumé

L'objectif du projet est d'implémenter différentes méthodes de résolution du problème de sac à dos binaire, afin des les comparer. Il s'agit des méthodes du Branch And Bound, de Programmation Dynamique (sous plusieurs implémentations), de Modèle de Flot dans un graphe, ainsi que l'utilisation du solveur Cplex. Le langage de programmation pour atteindre ce but est libre, mais sa rapidité est essentielle pour la résolution. J'ai par conséquent choisi de réaliser le projet en Java, au détriment de C++. Les résultats sont présentés en prenant en compte les comportements des algorithmes suivant les types d'instance (nombre d'objets, capacité maximale du sac, corrélation des objets). L'ensemble du projet peut se retrouver à l'adresse suivante : `https://github.com/LuminusDev/Knapsack/`.

Sommaire

1 Présentation du projet

1.1 Description du problème

Explication du problème de sac à dos, puis sac à dos binaire.

1.2 Méthodes de résolution

Pour résoudre ce problème, plusieurs algorithmes ont été implémenté.

— Branch And Bound
— Programme Dynamique (Simple Backward)
— Programme Dynamique (Forward avec liste)
— Programme Dynamique (Core avec liste)
— Plus court chemin dans un graphe

Un modèle du problème est aussi réalisé afin de résoudre les instances dans le solveur CPlex. Cela permet de vérifier les valeurs des solutions des algorithmes implémentés.

Le Branch And Bound utilise des bornes supérieures et inférieures pour couper des branches de l'arbre d'énumération des solutions, sur lequel on réalise un parcours en profondeur. Une borne supérieure est donné par la valeur de la solution partielle actuelle dans l'arbre, à laquelle on ajoute la valeur de la relaxation linéaire du problème partielle restant. Une borne inférieure correspond à la valeur de la meilleure solution complète trouvée à l'instant.

Programme Dynamique Simple backward.

Programme Dynamique Forward avec liste.

Programme Dynamique Core avec liste.

Plus court chemin graphe.

2 Implémentation des algorithmes de résolution du problème de sac à dos binaire

2.1 Branch And Bound spécialisé

2.2 Programme dynamique (simple backward)

2.3 Programme dynamique (forward en liste)

2.4 Programme dynamique (core en liste)

2.5 Plus court chemin dans un graphe

3 Résultats et analyses

Les instances sont créées à l'aide de deux générateurs :

— un générateur pseudo-aléatoire possédant comme paramètres le poids min et max d'un objet, ainsi que le profit min et max d'un objet,

— un générateur linéaire, avec un profit proportionnel au poids. Il permet de créer des instances fortement corrélées (tous avec le même ratio profit/poids par exemple).

Dans les résultats des jeux de données, tous les temps sont exprimés en millisecondes. De plus :

"OoM" : "Out Of Memory" correspondant à une trop grande utilisation de la mémoire (¿ 2Go).

"-" : temps d'exécution supérieur à 10 minutes.

3.1 Données faiblement corrélées

Le premier jeu de données est constitué d'instances générées aléatoirement, avec un poids inférieur à 25% de la capacité, et un profit inférieur à 50% de la capacité, sur 10 itérations.

	Bab			Backward			Forward			Core			Graphe		
(nbItem, cap)	mean	min	max	mean	min	max	mean	min	max	mean	min	max	mean	min	max
(200,400)	0	0	0	1	0	9	5	0	53	0	0	4	15767	13476	17309
(400,800)	0	0	1	4	0	27	7	0	58	1	0	7	254781	194129	279740
(800,1600)	0	0	1	9	0	19	6	0	48	2	0	9	-	-	-
(1600,3200)	0	0	0	14	0	47	15	0	62	0	0	0	-	-	-
(16K,32K)	0	0	1	OoM	OoM	OoM	50	15	110	19	0	62	OoM	OoM	OoM
(160K,320K)	18	15	47	OoM	OoM	OoM	266	109	594	128	62	289	OoM	OoM	OoM
(320K,640K)	36	5	54	OoM	OoM	OoM	503	227	1414	349	118	612	OoM	OoM	OoM
(640K,1280K)	101	26	166	OoM	OoM	OoM	994	613	1696	1527	257	2826	OoM	OoM	OoM
(1280K,2560K)	142	44	321	OoM	OoM	OoM	1030	630	2462	2098	469	5542	OoM	OoM	OoM

TABLE 1 – Résultats des données faiblement corrélées

A partir de 160K items, le core ne donne pas forcément la bonne solution (dû à l'utilisation du type "int" qui fausse le résultat de la relaxation linéaire sur de trop grand nombre).

On remarque que la méthode de **graphe** est extrêmement longue même avec de petites instances, avec 15 secondes en moyenne pour 200 objets et 400 de capacité, alors que tous les autres algorithmes donnent un résultat quasi instantané jusqu'à 1600 objets et 3200 de capacité. On peut l'expliquer par le fait que cette méthode ne permet pas de couper un ensemble de solution ou d'en éliminer par dominance.

De plus, les méthodes de **backward** et de **graphe** ne peuvent plus donner de résultat à partir de 16K objets à cause d'un dépassement de mémoire. La taille de la matrice de taille $N * W$ en backward et la taille du graphe (du même ordre de grandeur) limite la possibilité d'utilisation de ces algorithmes. Les temps en backward sont cependant bons lorsque la mémoire est suffisante, avec des mesures similaires aux autres programmes dynamiques (forward et core).

Les données faiblement corrélées permettent d'obtenir d'excellentes bornes et coupes par dominance. Ce qui explique des résultats toujours très bons pour le **branch-and-bound**, le **forward** et le **core**. On peut ainsi traiter des instances de très grandes tailles sans subir de réelle contreperformance (pour plus de 1M d'objets, on ne dépasse pas les 5 secondes dans le pire cas, et même 0.321 seconde pour le branch-and-bound).

3.2 Données très fortement corrélées

Le deuxième jeu de données est constitué d'instances générées de manière linéaire, avec un poids égal au profit (ex : (1,1)(2,2)(3,3)...), sur 10 itérations. Les ratios profit/poids sont ainsi identiques.

	Bab			Backward			Forward			Core			Graphe		
(nbItem, cap)	mean	min	max	mean	min	max	mean	min	max	mean	min	max	mean	min	max
(32,64)	0	0	1	0	0	2	6	0	51	3944	3874	4088	24	10	130
(33,66)	0	0	0	0	0	2	13	2	55	6076	5898	6180	40	13	137
(40,80)	0	0	0	0	0	2	36	13	55	81245	83478	84600	219	184	256
(50,100)	0	0	0	0	0	2	6	1	52	-	-	-	52	25	195
(10K,20K)	1	0	4	664	454	968	-	-	-	-	-	-	OoM	OoM	OoM
(100K,200K)	4	2	12	OoM	OoM	OoM	-	-	-	-	-	-	OoM	OoM	OoM
(10M,20M)	391	189	1940	OoM	OoM	OoM	-	-	-	-	-	-	OoM	OoM	OoM

TABLE 2 – Résultats

De la même manière que pour les instances non corrélées, les méthodes de graphe et de backward sont limités par leur utilisation excessive de la mémoire, ce qui ne permet pas de dépasser 10K objets et 100K objets respectivement.

On observe immédiatement un comportement de la méthode du **core** différent des autres méthodes, avec des temps dépassant les 3 secondes en moyenne, pour seulement 32 objets. En ajoutant seulement 1 objet, on augmente le temps moyen de 50%, alors que les autres algorithmes ne donnent pas de différence significatives. On pourrait expliquer ce comportement par le fait que la borne inférieure utilisée dans la méthode est légèrement inférieure à la solution optimale. L'algorithme réalise ensuite une recherche exhaustive des solutions, ne parvenant pas à supprimer des solutions que ce soit par borne ou par dominance. La méthode du **forward** n'arrive plus à suivre à partir d'un certain seuil de la même manière.

Le **branch-and-bound** permet de trouver des solutions très rapidement, car la première descente donne une borne qui est la solution optimale. L'algorithme résout ainsi ces instances en $\mathcal{O}(n^2)$, d'où des temps inférieur à la seconde pour des instances énormes (10M d'objets par exemple).

3.3 Données fortement corrélées

Le troisième jeu de données est constitué d'instances générées de manière linéaire, avec $poids(i) = minPoids + i$ et $profit(i) = minProfit + i * 0.2$, sur 10 itérations.

	Bab			Backward			Forward			Core			Graphe		
(nbItem, cap)	mean	min	max	mean	min	max	mean	min	max	mean	min	max	mean	min	max
(100,500)	0	0	0	1	0	5	6	0	53	34	25	73	5855	4468	6069
(200,1000)	0	0	0	2	0	11	5	0	53	5	2	18	74278	71470	76133
(1K,5K)	0	0	1	15	6	36	8	2	60	-	-	-	-	-	-
(10K,50K)	0	0	2	OoM	OoM	OoM	27	7	84	-	-	-	OoM	OoM	OoM
(100K,500K)	2	2	8	OoM	OoM	OoM	130	65	370	-	-	-	OoM	OoM	OoM
(220K,1100K)	35	20	80	OoM	OoM	OoM	-	-	-	-	-	-	OoM	OoM	OoM
(2M,10M)	281	149	697	OoM	OoM	OoM	-	-	-	-	-	-	OoM	OoM	OoM
(5M,25M)	390	354	531	OoM	OoM	OoM	-	-	-	-	-	-	OoM	OoM	OoM

TABLE 3 – Résultats

Les résultats de ce jeu de données ressemblent fortement au jeu de données précédent.

3.4 Données fortement corrélées sur le poids uniquement

Le quatrième jeu de données est constitué d'instances générées aléatoirement, avec $poids \in [nbItems/10, nbItems/10 + nbItems/100]$ et $profit \in [1, nbItems]$, sur 10 itérations.

	Bab			Backward			Forward			Core			Graphe		
(nbItem, cap)	mean	min	max	mean	min	max	mean	min	max	mean	min	max	mean	min	max
(200,1000)	6	0	52	3	0	16	6	0	49	1	0	5	70896	56038	73046
(400,2000)	3095	0	20178	4	1	16	23	1	82	8	0	44	-	-	-
(800,4000)	4280	0	31651	10	5	35	37	2	123	16	3	71	-	-	-
(1600,8000)	-	0	-	41	21	83	1284	1	2857	745	1	3432	-	-	-
(10K,50K)	-	-	-	OoM	OoM	OoM	19035	7712	31749	170000	170000	170000	-	-	-

TABLE 4 – Résultats

A partir de 160000 items, le core ne donne pas forcément la bonne solution (dû à l'utilisation du type "int" qui fausse le résultat de la relaxation linéaire sur de trop grand nombre).