Algoritmi de aproximare: analiza problema NP-complete
Problema Partitionarii (Partition Problem)

1 Introducere

1.1 Descrierea problemei de rezolvat:

Se da un set S de numere intregi pozitive si vrea sa se decida daca multimea poate fi impartita in doua submultimi S_1 si S_2 care sa aiba aceasi suma. Aceasta este una dintre cele 21 probleme originale pe care Richard Karp le-a demonstrat ca fiind NP-Complete. Este de asemenea una dintre cele 6 probleme fundamentale ale lui Garey si Johnson si singura problema NP-complete bazata pe numere.

Se da un set de intrare $S = \{s_1, s_2, s_3, ..., s_n\}$ de n intregi pozitivi. Se vrea impartirea in doua submultimi astfel incat suma submultimii mai mare sa fie minimizata.

$$minz = \sum_{a_i \in S_1} - \sum_{a_i \in S_2}$$

$S_1 \cup S_2 = n$
$S_1 \cap S_2 = \varnothing$
Definim:
S submultimea de numere
n numarul de elemente
k numarul de partitii (in cazul nostru k = 2)
P o partitie (va avea 2 submultimi)
C* cost optim
$C*_p$ costul unei partitii perfecte
sum(S) suma intregilor lui S

Daca exista o partitie unde suma submultimilor difera cu maxim 1, atunci o vom numi partitie perfecta si o notam cu P*.
Costul unei partitii perfecte:

$$cost(P*) = \frac{sum(S)}{2}$$

Problema descrisa are k = 2 si de aceea este un caz particular al problemei Multi-Way Number Partitioning. Ca aplicatie a acestei probleme avem Multiprocessor Scheduling. Numarul k corespunde cu numarul de core-uri identice ale unui procesor. Scopul acestei probleme este de a atribui fiecarui core al procesoului un numar din cele n joburi astfel incat timpul de efectuare a taskurilor sa fie minim.

1.2 Specificarea soltiilor alese:

Algoritmul Greedy (GA) Se sorteaza elementele din S in ordine descrescatoare dupa care se adauga cate un element pe rand in fiecare submultime in functie de submultimea cu suma mai mica. Daca cele doua sume sunt egale atunci se alege arbitrar una dintre ele. Solutia are complexitatea timp O(nlong n) si cea spatiu O(n).

Diferenta de submultimi: The Karmarkar-Karp Algorithm (KK) Se sorteaza elementele din S in ordine descrescatoare dupa care se inlocuieste cele mai mari 2 numere cu diferenta dintre ele. Aceasta este echivalent cu a pune cele doua elemente in multimi separate fara a specifica in care multime. Algoritmul continua pana cand ramane un sigur intreg care este diferenta dintre multimile S1 si S2.

1.3 Criteriile de evaluare pentru solutia propusa:

Ca o prima metoda de testare voi alege intregi random din intervalul $[1, 2^{32} - 1]$. Generam 100 de probleme pentru:
- n-ul variaza de la 0 la 5000
Inregistram o medie a runtime-ului pentru fiecare din cei 2 algoritmi.

Ca o a doua metoda de testare voi alege intregi random din intervalul $[1, 2^{24} - 1]$. Alegem intregi de 24 de biti astfel incat sa obtinem un test mai greu care nu are partitii perfecte. Ultimele 19 de teste au elemente unice.
Generam 100 de probleme pentru:
- n-ul variaza de la 0 la 5000
Inregistam o medie a runtime-ului pentru fiecare din cei 2 algoritmi.
De asemenea vom inregistra procentul de partitii perfecte din cele 100 generate pentru fiecare test.

2 Prezentare solutiilor:

Ca metoda de implementare ai ambilor algorimi am ales sa folosesc liste dublu inlantuite. Am implementat o serie de functii pentru a lucra mai usor.

2.1 Algorimul Greedy

In cazul algorimului Greedy, implementarea este destul de simpla. Functia primeste o lista dublu inlantuita sortata. Pe aceasta lista se face pop la inceput si se aduna valoarea la suma cea mai mica. La sfarsit vom avea doua sume, S_1 si S_2, care am

vrea noi sa fie egale. Functia returneaza diferenta pe care am evaluat-o ulterior. Complexitatea algoritmului este O(n). Exemplu:

$$S = \{18, 17, 12, 11, 8, 2\}$$

1.$S_1 = 18, S_2 = 17$
2.$S_1 = 18, S_2 = 29$
3.$S_1 = 29, S_2 = 29$
4.$S_1 = 37, S_2 = 29$
5.$S_1 = 37, S_2 = 31$

Diferenta in acest caz este 6.

$$Costul = (sum(S) + diferenta)/2 = (37 + 6)/2 = 21$$

corespunzator partitiilor $< \{18, 11, 8\}, \{17, 12, 2\} >= \{37, 31\}$

2.2 Algoritmul Karmarkar-Karp

Algoritmul Karmarkar-Karp este mai eficent decat abordarea Greedy. Functia primeste aceeasi lista dublu inlantuita ca si algoritmul anterior. La fiecare pas se selecteaza primele 2 valori de la inceputul listei care sunt si valorile cele mai mari. Se face diferenta pe aceste doua valori, diferenta care se insereaza mai apoi in lista in mod sortat. La final va ramane un sigur numar care este si diferenta dintre sumele celor doua multimi. Algoritmul returneaza diferenta, iar in cazul in care ne intereseaza si cum arata cele doua multimi va trebui sa facem niste operatii in plus. Cel mai usor acest lucru poate fi implementat cu un graf.
Implementarea fiind realizata cu liste, avem o complexitatea de n^2 intrucat stergem elemente si inseram intr-o lista ordonata. Chiar daca algorimul Karmarkar-Karp are in mare parte rezultate mai bune decat abordarea Greedy, exista totusi cazuri in care algorimul se comporta mai prost.
Exemplu:

$$S = \{18, 17, 12, 11, 8, 2\}$$

1. $18 - 17 = 1$, stergem primele doua si inseram 1

$$\{12, 11, 8, 2, 1\}$$

2. $12 - 11 = 1$, stergem primele 2 si inseram 1

$$\{8, 2, 1, 1\}$$

3.$8 - 2 = 6$, stergem primele doua si adaugam 6

$$\{6, 1, 1\}$$

4. $6 - 1 = 5$, stergem primele 2 si inseram 5

$$\{5, 1\}$$

5. $5 - 1 = 4$, stergem primele 2 si inseram 4

$$\{4\}$$

6. 4 este ultimul element; rezulta ca 4 este diferenta dintre cele doua partitii

$$Costul = (sum(S) + diferenta)/2 = 36$$

corespunzator partitiilor $< \{18, 12, 2\}, \{17, 11, 8\} >= \{32, 36\}$

In general algoritmul Karmarkar-Karp depasteste algoritmul Greedy cand vine vorba de calitatea solutiei. Pentru ambii algoritmi, diferenta partitiei finale e de ordinul marimii ultimului element care trebuie atribuit. Pentru Greedy e cel mai mic numar din multimea originala. Pentru KK, facand diferenta continuu reduce dimensiunea numarul final care trebuie atribuit.

*Complexitatea ambilor algorimi este considerata fara sortare. Daca se considera si sortarea atunci primul algoritm ar avea complexitatea sortarii (+ n pasi la pop care dau in final complexitatea sortarii). Daca se foloseste quicksort atunci vom avea $n * log(n)$. La al doilea algoritm se adauga si complexitatea sortarii dar in final tot $O(n^2)$ ajunge sa fie. Complexitatea spatiu, este identica la ambii algoritmi si este egala cu O(n).

3 Evaluare:

3.1 Constructia setului de teste

Testele au fost facute cu ajutorului unu program scris in Java care genereaza N numere intr-un anumit interval specificat. Programul genereaza numerele care sunt scrise intr-un fisier sub forma:
- primul element este numarul de elemente generate
(elementele multimii)
- urmatoarele numere sunt elementele multimii
*toate numerele sunt delimitate de spatiu

3.2 Citirea din fisier si cum functioneaza programul principal

Se citeste primul element din fisier care se stie ca e dimensiunea multimii. Se ia fiecare element pe rand si se insereaza intr-un vector. Vectorul se sorteaza. Sortarea se face cu qsort. Se ia fiecare element din lista formeaza o lista. Lista va fi trimisa mai departe la functii.

3.3 Rezultatele

Testele au fost rulate pe un sistem cu procesor Intel i7-8550U cu frecventa de 1.8GHz. Pentru a realiza lucrarea am folosit si niste scripturi facute de mine atat in C cat si in Matlab.

Fiecare problema cu doua partitii are 2^n partitii complete. Daca intregii sunt luatii din intervalul $[1, 2^b - 1]$ unde b este numarul de biti pe care este reprezentat intregul de input atunci sunt maxim n x $(2^b - 1)$ posibile sume.
Pentru un b fixat, numarul de partitii complete creste exponential cu N in timp ce numarul de sume unice creste liniar. Partitiile cu sume extreme sunt rare. Daca b e mic si N e mare atunci vor fii mai multe partitii complete decat sume unice posibile pentru fiecare parititie facand sansele de a gasi o solutie destul de mari.
Daca o partitie exista atunci orice algoritm de partitionare o poate returna ca optima.
Cand partitia perfecta exista, raportul de partitii perfecte e cel care determina dificultatea problemei in cauza. Cand nu exista partitie perfecta cu cat n creste, problema tinde sa devina mai dificila, desi intr-un punct N, partitiile perfecte incep sa apara si problemele devin mai usoare din nou intrucat numarul de partitii perfecte creste exponential.

3.4 Set 1.

Testele **0 - 49**;
Interval generare: 1 - 200 cu dubluri;
Toate multimile au 2000 de numere generate.

Rezultate Greedy:
Diferenta de sume = 0: **23/50**
Diferenta de sume = 1: **23/50**
Diferenta de sume <= 100: **50/50**
Rezultate KK:
Diferenta de sume = 0: **24/50**
Diferenta de sume = 1: **26/50**
Diferenta de sume <= 100: **50/50**

3.5 Set 2.

Testele **50 - 100**;
Interval de generare: 1-2147483647 cu dubluri;
Numarul de elemente din multime: Random intre [1-5000].

Rezultate Greedy:
Diferenta de sume = 0: **2/50**
Diferenta de sume = 1: **9/50**
Diferenta de sume <= 100: **20/50**
Rezultate KK:
Diferenta de sume = 0: **24/50**
Diferenta de sume = 1: **20/50**
Diferenta de sume <= 100: **45/50**

3.6 Set 3.

Testele **100 - 180**;
Interval de generare: 16777216 - 33554432 cu dubluri;
Numarul de elemente din multime: Random intre [1-5000].

Rezultate Greedy:
Diferenta de sume = 0: **0/50**
Diferenta de sume = 1: **0/50**
Diferenta de sume <= 100: **0/50**
Rezultate KK:
Diferenta de sume = 0: **16/50**
Diferenta de sume = 1: **30/50**
Diferenta de sume <= 100: **46/50**

3.7 Set 4.

Testele **181 - 200**;
Interval de generare: 1-2147483647 fara dubluri;
Numarul de elemente din multime: 2000.
Rezultate Greedy:
Diferenta de sume = 0: **0/50**
Diferenta de sume = 1: **0/50**
Diferenta de sume <= 100: **0/50**
Rezultate KK:
Diferenta de sume = 0: **8/50**
Diferenta de sume = 1: **12/50**
Diferenta de sume <= 100: **20/50**

3.8 Runtime

Medie RUNTIME Greedy: 30 ms
Medie RUNTIME KK: 39005 ms

Rezultatul timpului la runtime este de asteptat intrucat algoritmul KK trebuie sa insereze sortat in lista, aceasta operatie fiind destul de costisitoare. Observam de asemenea faptul ca Greedy are un timp bun chiar daca rezultatele sunt mai slabe ca KK.

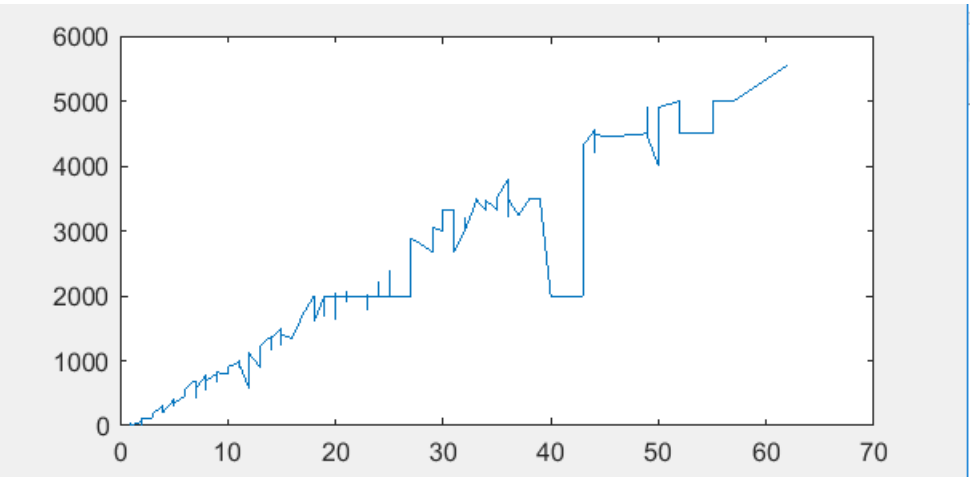

Fig. 1. Greedy Runtime

In urma testelor aplicate celor doi algoritmi am creat doua tabele unde este contorizat pe fiecare set de teste numarul de aparitii a urmatoarelor cazuri:
1.Diferenta de sume = 0
2.Diferenta de sume = 1
3.Diferenta de sume < 100

$AlgoritmulGreedy(GA)$

Interval	NumOf $S = 0$	NumOf $S = 1$	NumOf $S < 100$
[0, 49]	23	23	50
[50, 100]	2	9	20
[101, 180]	0	0	0
[181 - 200]	0	0	0

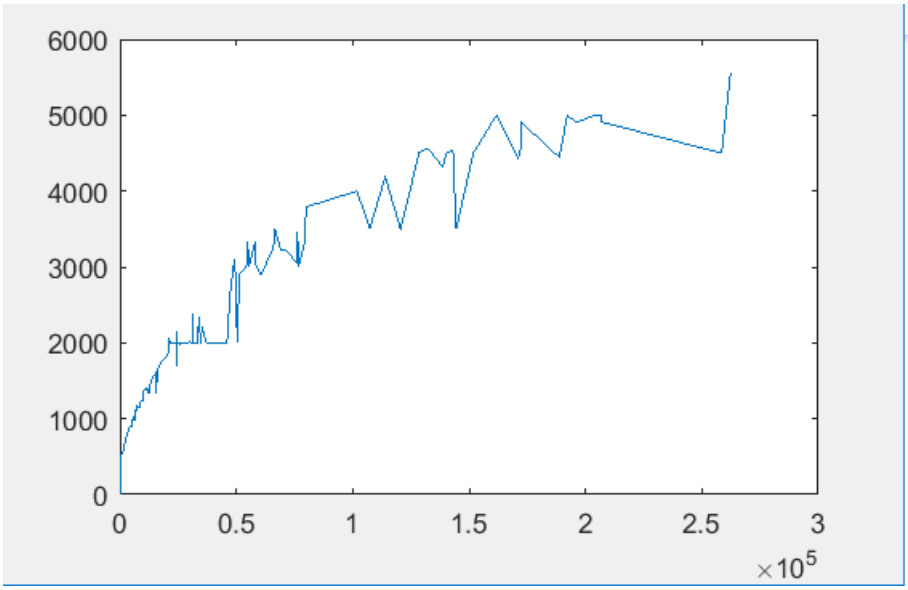

Fig. 2. KK Runtime

$$AlgoritmulKarmarkar - Karp(KK)$$

Interval	NumOf $S = 0$	NumOf $S = 1$	NumOf $S < 100$
[0, 49]	24	26	50
[50, 100]	24	20	45
[101, 180]	16	30	46
[181 - 200]	8	12	20

Observam faptul ca odata ce am crescut intervalul de generare, algoritmul greedy nu reuseste sa mai genereze partitii perfecte.
Algoritmul KK observam ca are un comportament bun si pe ultimele 19 teste care sunt cele mai grele.

4 Concluzie

In urma analizei am conclus ca in practica pot fi folositi ambii algorimi in functie de nevoie chiar daca la prima vedere algorimul KK pare superior. Timpul de rulare algorimului Karmarkar-Karp este mult mai mare fata de a abordarii Greedy deci in cazul in care ma intereseaza o solutie rapida si nu exacta pe o multime mare de numere pot folosi Greedy. Daca imi trebuie insa o metoda care in balanseaza partitiile mai calitativ as folosi KK chiar daca acesta dureaza mai

mult.

4.1 Important

*Arhiva pe care am lucrat e putin modificata fata de cea de la pasul 2 intrucat am observat niste greseli pe care le-am corectat. Daca o trimiteam din nou eram depunctata. Ideea e ca studiul e facut pe algorimii corecti si pe arhiva corecta.

References

1. Zuckerman, David. "NP-complete problems have a version that's hard to approximate." Structure in Complexity Theory Conference, 1993., Proceedings of the Eighth Annual. IEEE, 1993.
2. Schreiber, Ethan L. Optimal Multi-Way Number Partitioning. Diss. UCLA, 2014.
3. Ducha, Fernando Andrade, and Srgio Ricardo de Souza. "Algorithms analysis for the number partition problem." XXXIV CILAMCE (2013).
4. Hwang, F. K. "Optimal partitions." Journal of Optimization Theory and Applications 34.1 (1981): 1-10.