Fakulta riadenia a informatiky
Informatika

H16
Primárna heurestika s výhodnostnými koeficientami

doc. Ing. **Michal Kohání,** PhD.
UTOROK 11, 12
2021/2022

Maroš Gorný, 5ZYI21

Obsah

Zadanie H16

Primárnou heuristikou s výhodnostnými koeficientmi riešite úlohu danú modelom (obrátená úloha o batohu, kde hmotnosť batohu musí byť aspoň *K* a počet predmetov v batohu aspoň *r*). Riešte úlohu pre *n*=500, *r*=350, *K*=10500 a pre **lokálne kritérium** *„Odstráň prvokz dosiaľ nespracovaných prvkov, ktorý* ***má najväčší pomer koeficientov*** c_j **/**a_j *(najväčší výhodnostný koeficient)*“. Východiskové riešenie položte rovné batohu, v ktorom sú vložené všetky prvky. Súčasťou zadania sú súbory **H6_a.txt** a **H6_c.txt**, ktoré obsahujú *n* údajov koeficientov a_j a c_j pre *j*=1..*n* potrebných pre riešenie zadanej úlohy.

$$\text{Min} \quad \sum_{j=1}^{n} c_j z_j$$

$$\text{za podmienok} \quad \sum_{j=1}^{n} z_j \geq r$$

$$\sum_{j=1}^{n} a_j z_j \geq K$$

$$z_j \in \{0,1\} \ \forall j=1..n$$

Popis riešeného algoritmu na konkrétnej úlohe

Úvod

Na začiatok je potrebné **upresniť úlohu** a určiť cieľ aby som presne vedel, že čo mám robiť, akým postupom a za akých podmienok sa ku tomu cieľu dostanem.

Cieľ

Cieľ pri zadanej úlohe je účelová funkcia

$$min \sum_{j=1}^{n} c_j z_j$$

Kde **minimalizujeme** cenu v batohu

za podmienok

$$\sum_{j=1}^{n} z_j \geq r$$

Počet prvkov v batohu musí byť **väčší alebo rovný ako** r**.**

$$\sum_{j=1}^{n} a_j z_j \geq K$$

Hmotnosť všetkých prvkov v batohu musí byť **väčšia alebo rovná ako K**.

$$z_j \in \{0,1\} \; \forall \; j = 1..n$$

Z_j môže nadobúdať len hodnoty **0 a 1**, pre všetky prvky danej úlohy.
0 znamená, že prvok v batohu nie je, naopak 1 znamená, že prvok v batohu je.

lokálne kritérium

„Odstráň prvok z dosiaľ nespracovaných prvkov, ktorý ***má najväčší pomer koeficientov*** c_j ***/*** a_j *(najväčší výhodnostný koeficient)“.*

To znamená, že z batohu budem postupne odoberať prvky ktoré má najväčší výhodnostný koeficient .

Východiskové riešenie

„Východiskové riešenie položte rovné batohu, v ktorom sú vložené všetky prvky.“
To znamená, že úlohu začnem riešiť s plným batohom, teda v batohu budú všetky prvky. Týmto postupom sme už na začiatku dostali prípustné riešenie, ktoré sa ale budeme snažiť zlepšiť.

Vysvetlivky

n *je počet všetkých predmetov,*
r *je minimálny počet predmetov v batohu,*
K *je minimálna hmotnosť akú musí mať batoh,*
c_j *je cena j-teho predmetu,*
a_j *je hmotnosť j-teho predmetu,*
z_j *je j-ty predmet, 1 ak je vložený, 0 ak vložený nie je.*

Postup riešenia

1. Určím si kritérium
    a. Prvok ktorý vložím do batohu musí mať maximálny výhodnostný koeficient z doposiaľ nevložených prvkov.
2. Vypočítam si výhodnostný koeficient pre každý prvok
3. Určím si počet všetkých prvkov
    a. Počet všetkých prvkov je 500.
4. Všetky prvky vložím do batohu.
5. Určím si podmienky
    a. Prvok môže byť len vložený alebo nevložený v batohu (1/0).
    b. Počet prvkov v batohu musí byť aspoň 350.
    c. Hmotnosť batohu musí byť aspoň 10 500.
6. Vyberiem prvok z batohu, podľa kritéria a za daných podmienok
    a. Ak už nemôžem vybrať ďalší predmet bez porušenia podmienok, skočím na krok 7
    b. Ak sú splnené všetky podmienky, opakujem krok 6
7. Končím s algoritmom a účelová funkcia sa rovná sume ceny všetkých prvkov v batohu.

Popis jednotlivých tried programu

V programe sa nachádzajú dve triedy, trieda **Main** a trieda **HeurestikaH16**

Main()

Trieda ***Main()*** slúži na spustenie programu, vytváram v nej objekt ***Heuristika16*** s názvom *„heurestika“*, ktorý si vypýta dva textové súbory: prvý je textový súbor, v ktorom je vložená hmotnosť prvkov a druhý je textový súbor, v ktorom je vložená cena prvkov. Následne len vyvolávam metódy triedy Heuristika16.

Zdrojový kód triedy Main()

```
import java.io.FileNotFoundException;

public class Main {
    public static void main(String[] args) throws FileNotFoundException {

        HeuristikaH16 heurestika;
        heurestika = new HeuristikaH16("H6_a.txt","H6_c.txt");
        heurestika.primarnaVyhodnostnyKoeficient();

        heurestika.vypisVysledkov();
        heurestika.zapisVysledokDoSuboru("Výstup_H16 - pôvodná úloha.txt");
    }
}
```

HeurestikaH16

Trieda HeurestikaH16 slúži na vykonávanie všetkých procesov ohľadom algoritmu pre danú heurestiku, na výpis na konzolu alebo na zápis a čítanie z alebo do textového súboru.

Premenné

Premenné slúžia na uchovanie dôležitých informácií alebo na uchovanie často používaných konštánt.

```
private int[] a;
private int[] c;
private double[] koeficient;
private boolean[] z;
private int aktualnaHmotnost;
private int aktualnaCena;
private int aktualnyPocetPrvkov;

private static final int n = 500;
private static final int r = 350;
private static final int K = 10500;
```

a – hmotnosť
c – cena
koeficient – pomer cena/hmotnosť
z – vybraný/nevybraný prvok
aktualnaHmotnost – aktuálna hmotnosť batohu
aktualnaCena – aktuálna cena batohu
aktualnyPocetPrvkov – aktuálny počet prvkov v batohu
n – počet prvkov v súbore
r – minimálny počet prvkov v batohu
K – minimálna hmotnosť v batohu

Konštruktor

Konštruktor slúži na inicializáciu premenných a nastavenie vstupných súborov.

```
public HeuristikaH16(String nazovSuboruHmotnost, String nazovSuboruCena)
throws FileNotFoundException {

    a = new int[n];
    c = new int[n];
    z = new boolean[n];
    koeficient = new double[n];

    Arrays.fill(z,true);
    nacitajZoSuboruHmotnost(nazovSuboruHmotnost);
    nacitajZoSuboruCenu(nazovSuboruCena);

    for (int i = 0; i < n; i++) {
        koeficient[i] = c[i]/a[i];
    }

    aktualnaCena = Arrays.stream(c).sum();
    aktualnaHmotnost = Arrays.stream(a).sum();
    aktualnyPocetPrvkov = n;
}
```

a – inicializácia hmotnosti podľa prvku pomocou metódy „nacitajZoSuboruCenu(nazov)“
c – inicializácia cien podľa prvku pomocou metódy „nacitajZoSuboruCenu(nazov)“
z – inicializácia na TRUE hodnoty
koeficient – inicializácia na výhodnostné koeficienty podľa prvkov
aktualnaCena – inicializácia na súčet všetkých cien v batohu
aktualnaHmotnost – inicializácia na súčet všetkých hmotností v batohu
aktualnyPocetPrvkov – inicializácia na počet prvkov

Metódy

public void vypisVysledkov()

Metóda určená na vypísanie výsledku na terminál. Ktorý bude vyzerať takto.

Aktuálna cena batohu: XXX
Aktuálna hmotnosť: XXX
Aktuálny počet prvkov: XXX

Hodnota účelovej funkcie: XXX

Index vybraného predmetu... (1 až 500)
X,XX,XX,XX,XXX,XXX..

```
public void vypisVysledkov() {
    System.out.println("Aktuálna cena batohu:\t" + aktualnaCena);
    System.out.println("Aktuálna hmotnosť:\t\t" + aktualnaHmotnost);
    System.out.println("Aktuálny počet prvkov: \t" + aktualnyPocetPrvkov);
    System.out.println();
    System.out.println("Hodnota účelovej funkcie:\t" + aktualnaCena);

    System.out.println();
    System.out.println("Index vybraného predmetu... (1 až " + n + ")");
    for (int i = 0; i < n; i++) {
        if ((i + 1) % 100 == 0) {
            System.out.println();
        }
        if (z[i] == true) {
            System.out.print(i + 1 + ", ");
        }
    }
}
```

public int getIndexNajvacsiKoeficientZaDanychpodmienok()

Metóda určená na vyhľadávanie najväčšieho koeficientu za daných podmienok. Teda vyhľadá taký, najväčší koeficient ktorý je možné odstrániť z batohu.

Kontroluje sa tam podmienka či je daný koeficient ešte v batohu a či po vybraní daného prvku neklesnem pod podmienku s hmotnosťou.

```
public int getIndexNajvacsiKoeficientZaDanychpodmienok(){
    double docasneMaximum = -1;
    int indexMaxima = -1;

    for (int i = 0; i < n; i++) {
        if (koeficient[i] > docasneMaximum && z[i] == true) {
            if ((aktualnaHmotnost - a[i]) >= K) {
                docasneMaximum = koeficient[i];
                indexMaxima = i;
            }
        }
    }
    return  indexMaxima;
}
```

public void odstranPrvokPodlaKoeficientu(int indexPrvku)

Metóda ktorá mi odstráni prvok z batohu podľa daného koeficientu.
To znamená, že označím prvok, že som ho vybral a odpočítam cenu a hmotnosť daného prvku z batohu.

```
public void odstranPrvokPodlaKoeficientu(int indexPrvku) {
    z[indexPrvku] = false;
    aktualnaHmotnost -= a[indexPrvku];
    aktualnaCena -= c[indexPrvku];
    aktualnyPocetPrvkov --;
}
```

public void primarnaVyhodnostnyKoeficient()

Metóda v ktorej sa snažím 500 krát prejsť celý zoznam prvkov a vždy vybrať ten s najväčším výhodnostným koeficientom a odstrániť ho z batohu.

Ak už žiadny koeficient ktorý by som mohol vybrať nenájdem, cyklus končí.

```
public void primarnaVyhodnostnyKoeficient() {
    for (int i = 0; i < n; i++) {
        int indexNajvacsiehoKoeficientu =
getIndexNajvacsiKoeficientZaDanychpodmienok();

        if (indexNajvacsiehoKoeficientu == -1) {
            break;
        }

        if (aktualnyPocetPrvkov > r) {
            odstranPrvokPodlaKoeficientu(indexNajvacsiehoKoeficientu);
        } else {
            break;
        }
    }
}
```

public void nacitajZoSuboruHmotnost(String nazovSuboru)

Metóda ktorá slúži na načítanie hmotnosti do batohu (pola).

```
public void nacitajZoSuboruHmotnost(String nazovSuboru) throws
java.io.FileNotFoundException
{
    java.util.Scanner citac = new java.util.Scanner(new
java.io.File(nazovSuboru));
    for (int i=0; i<n; i++) {
        a[i]=citac.nextInt();
    }
    citac.close();
}
```

public void nacitajZoSuboruCenu(String nazovSuboru)

Metóda ktorá slúži na načítanie ceny do batohu (pola).

```
public void nacitajZoSuboruCenu(String nazovSuboru) throws
java.io.FileNotFoundException
{
    java.util.Scanner citac = new java.util.Scanner(new
java.io.File(nazovSuboru));
    for (int i=0; i<n; i++) {
        c[i]=citac.nextInt();
    }
    citac.close();
}
```

public void zapisVysledokDoSuboru(String nazovSuboru)

Metóda ktorá zapíše výsledok do súboru ktorý bude vyzerať takto:

Počet predmetov = XX
Hmotnosť batohu = XX
Účelová funkcia = XX

Index prvku	Hmotnosť	Cena
X	XX	XX
X	XX	XX

```
public void zapisVysledokDoSuboru(String nazovSuboru) {

    try {
        FileWriter zapis = new FileWriter(nazovSuboru);
        zapis.write("Počet predmetov = " + aktualnyPocetPrvkov + "\nHmotnosť
batohu = " + aktualnaHmotnost + "\nÚčelová funkcia = " + aktualnaCena +
"\n\n");
        zapis.write("Index prvku\tHmotnosť\tCena\n");
        for (int i = 0; i < n; i++) {
            if (z[i] == true) {
                zapis.write(String.valueOf(i + 1) + "\t\t" + a[i] + "\t\t" +
c[i] + "\n");
            }
        }
        zapis.close();
    } catch (IOException e) {
        System.out.println("Vyskytla sa chyba!");
        e.printStackTrace();
    }
}
```

Záverečné vyhodnotenie

Účelová funkcia

$$min \sum_{j=1}^{n} c_j z_j = 30395$$

Konečný stav batohu

Aktuálna cena batohu:	30395
Aktuálna hmotnosť:	21731
Aktuálny počet prvkov:	350

Zhodnotenie

Daná heuristika nám zabezpečila cenu batohu rovnej 30395, pričom prvkov bolo 350 a hmotnosť batohu bola 21731.

Výsledok je prípustný, ale určite nie je najoptimálnejší. Vzhľadom na to, že náš cieľ bol minimalizovať cenu za daných podmienok, tento algoritmus raz môže byť lepší a pri iných prvkoch môže byť naopak horší.

Pokiaľ by napríklad platilo to, že prvky s najväčším výhodnostným koeficientom by mali veľkú hmotnosť, mohlo by sa stať, že by sme prešli pod náš limit hmotnosti ešte s veľkým počtom prvkov v batohu a teda mohli by sme tam nájsť predmety, ktoré by sme vedeli využiť efektívnejšie.

V danom prípade sme mohli spotrebovať ešte 11 731 hmotnosti. Nemusí to znamenať to, že sme mohli vyťahovať prvky efektívnejšie, ale rozhodne je tu šanca, že je tu aj lepšie a výhodnejšie riešenie.

Bonusová úloha – vylepšenie heuristiky

Zadanie bonusovej úlohy

Navrhnite a implementujte vlastný heuristický algoritmus na zlepšenie výsledkov zadanej heuristiky. Algoritmus a dosiahnuté výsledky popíšte v dokumentácii.

Popis riešeného algoritmu na konkrétnej úlohe

Úvod

Vylepšený algoritmus som sa rozhodol robiť na pôvodnej úlohe. Rozhodol som sa teda konečný výsledok zadania **H16** vylepšiť, bez toho, aby som menil pôvodne zadanie a pridal algoritmus, ktorý mi vymení prvky z batohu tak, aby bola účelová funkcia menšia.

Cieľ

Cieľ je teda skúsiť vymeniť každý prvok, ktorý je v batohu s prvkom ktorý v batohu nie je, tak aby boli stále splnene podmienky a aby cena nového prvku bola menšia ako cena starého prvku.

Účelová funkcia, podmienky, lokálne kritérium a východiskové riešenie sú rovnaké ako v pôvodnej úlohe.

Avšak keď dosiahnem výsledok z prvej heurestiky, spúšťam nový algoritmus, ktorý mi začne vymieňať prvky tak, aby sa pokúsil účelovú funkciu zmenšiť.

Postup riešenia

Začiatok postupu riešenia je taký istý ako v pôvodnej úlohe a následne sa zavolá vylepšenie algoritmu ktoré má takéto kroky:

1. Vyberiem si n-ty prvok z batohu n=(1..500)
2. Nájdem ďalší prvok ktorý nasleduje po n-ku a nie je v batohu
    a. Ak taký prvok nenájdem skočím na krok 8
3. Skúsim ho vymeniť
    a. Ak sú aj po výmene splnené všetky podmienky a nová cena je menšia ako pôvodná cena, tak skočím na krok 4
    b. Ak by po výmene neboli splnené podmienky, alebo by nová cena bola väčšia alebo rovná ako pôvodná cena, tak skočím na krok 2
4. Ak som prešiel celý zoznam, idem na krok 5,
    a. Inak: Zapamätám si vybraný prvok a skočím na krok 2
5. Zoberiem si posledný prvok, ktorý som si zapamätal
    a. Ak som si nezapamätal žiadny prvok, inkrementujem n-ko a skočím na krok 2
    b. Ak som si zapamätal nejaký prvok, skočím na krok 6
6. Vyhodím n-ty prvok z batohu a posledný prvok ktorý som si zapamätal vložím do batohu.
7. Aktualizujem cenu a hmotnosť batohu, vymažem si posledný zapamätaný prvok z batohu.
8. Zväčším n-ko o jedna.
9. Ak n = 500, algoritmus sa končí, inak skočím na krok 1

Metóda na vylepšenie heurestiky

Metóda sa snaží vymeniť každý vložený prvok, s každým prvkom ktorý nie je v batohu, za podmienky, že sa zmenší cena batohu a že zostanú splnené aj základne podmienky.

```
public void vylepsenieHeurestiky() {
    int indexNahradnehoPrvku = -1;
    int novaCena = aktualnaCena;
    int novaHmotnost = aktualnaHmotnost;


    for (int i = 0; i < n; i++) {
        if (z[i] == true) {
            for (int j = 0; j < n; j++) {
                if (j == i || (z[j] == true)) continue;
                int hmotnostPoVymene = aktualnaHmotnost - a[i] + a[j];
                int cenaPoVymene = aktualnaCena - c[i] + c[j];
                if (hmotnostPoVymene >= K && cenaPoVymene < aktualnaCena) {
                    if (cenaPoVymene > novaCena) continue;
                    indexNahradnehoPrvku = j;
                    novaCena = cenaPoVymene;
                    novaHmotnost = hmotnostPoVymene;
                }
            }
            if (indexNahradnehoPrvku != -1) {
                z[i] = false;
                z[indexNahradnehoPrvku] = true;
                aktualnaHmotnost = novaHmotnost;
                aktualnaCena = novaCena;

                indexNahradnehoPrvku = -1;
            }
        }
    }
}
```

Účelová funkcia

$min \sum_{j=1}^{n} c_j z_j$ = 29085

Konečný stav batohu

Aktuálna cena batohu: 29085
Aktuálna hmotnosť: 20483
Aktuálny počet prvkov: 350

Zhodnotenie

Daná heuristika nám zabezpečila cenu batohu rovnej 29085, pričom prvkov bolo 350 a hmotnosť batohu bola 20483.

Zlepšenie oproti základnému zadaniu je:
Účelová funkcia (cena batohu) sa zmenšila o 1310 jednotiek.
Hmotnosť batohu sa zmenšila o 1248 jednotiek.

Výsledok je prípustný a tiež je **lepší ako predošlé riešenie**. Môže sa ale stať, že sa daná heurestika **zasekne v lokálnom minime** a tým pádom sa už nebude dať zlepšiť aj keď by tam priestor na zlepšenie bol.

Aj napriek tomu, že vylepšenie sa mi môže zaseknúť v lokálnom minime, tak moje riešenie považujem za **vhodné** a **prijateľné vylepšenie** pre heurestiku.

Najlepšie riešenie by som dostal **kombináciou všetkých prvkov**. To by mi ale rýchlostná **zložitosť** rástla **exponenciálne** a pri väčších úlohách by výpočet trval buď veľmi dlho, alebo by som ho ani nedostal..

V **heurestikách** však ide o to, aby som **rýchlo** a **jednoducho** dostal čo **najlepšie prístupné riešenie** a to dané vylepšenie spĺňa.