Kratka predstavitev robustnega problema nahrbtnika

Eva Babnik in Jan Založnik

November, 2020

1 Problem nahrbtnika

Spodobi se, da v uvodu najprej predstaviva klasični problem nahrbtinka (angl. *classical Knapsack Problem*) v nadaljevanju poročila bova uporabljala kar kratico KP. Na voljo imamo množico n-tih predmetov, ki jo označimo z $N = \{1, \ldots, n\}$ in nahrbtnik s kapaciteto c. Vsak predmet ima pozitivno vrednost p_j in pozitivno utež w_j. Problem nahrbtnika nas sprašuje, katero podmnožico predmetov iz N moramo položiti v nahrbtnik, da bo skupna vrednost le teh čimvečja možna in da ne bo presegla nahrbtnikove kapacitete. Torej maksimiziramo skupno vrednost predmetov pri pogoju, da seštevek izbranih uteži ne presega nahrbtinkove zmogljivosti. Problem lahko predstavimo tudi kot celoštevilski linearni program (CLP):

$$max \sum_{j \in N} p_j x_j$$
$$\sum_{j \in N} w_j x_j \leq c$$
$$x_j \in \{0, 1\}, j \in N$$

Kjer x_j zavzame vrednost 1, če j-ti predmet položimo v nahrbtnik, sicer zavzame vrednost 0.

2 Robustni problem nahrbtnika

Robustni problem nahrbtnika (angl. *Robust Knapsack Problem*) v nadaljevanju RKP, je nekakšna nadgranja problema nahrbtnika. Dodaten problem se pojavi pri točnosti naših podatkov, in sicer pri utežeh w_j. Vsak predmet j ima svojo nominalno težo w_j, ki pa je lahko netočna, ampak zanjo vemo, da se nahaja na intervalu $[w_j - \underline{w}_j, w_j + \overline{w}_j]$. Podan imamo tudi celoštevilski parameter Γ, ki označuje največje možno število predmetov z netočno izmerjeno težo. Pri iskanju rešitve problema moramo torej paziti, da bo seštevek vseh novih uteži še vedno manjši od kapacitete nahrbtnika. Težav z rešitivijo seveda ne bomo imeli, če bodo vse dejanske uteži nižje oziroma lažje od njene nominalne vrednosti, lahko pa se zgodi tudi najslabši možni primer, ko vse uteži zavzamejo zgornjo mejo intervala. Dopustno rešitev, kjer je $J \subseteq N$ lahko formuliramo na naslednji način:

$$\sum_{j \in J} w_j + \sum_{j \in \hat{J}} \overline{w}_j \leq c, \quad \forall \hat{J} \subseteq J \text{ in } |\hat{J}| \leq \Gamma$$

3 PRISTOP Z DINAMIČNIM PROGRAMIRANJEM

Naj bo $\bar{z}(d, s, j)$ najvišji dobiček za dopustno rešitev s skupno težo d, kjer so upoštevani samo elementi iz množice $\{1, \ldots, j\} \in N$ in samo s izmed njih doseže zgornjo mejo $\hat{w}_j$. Naj bo $z(d, j)$ največji dobiček za dopustno rešitev s skupno težo d, kjer so upoštevani samo elementi iz množice $\{1, \ldots, j\} \in N$ in naj jih samo Γ spremeni težo iz nominalne na zgornjo mejo $\hat{w}_j$. Torej velja $d = 0, 1, ..., c;\ s = 0, 1, \ldots, \Gamma$ in $j = 0, 1, \ldots, n$. Ključna lastnost pravilnosti tega pristopa je predpostavka, da so predmeti razvrščeni po padajoči teži $\hat{w}_j$. Problem lahko zapišemo z naslednjimi rekurzivnimi zvezami:

$$\bar{z}(d, s, j) = max\{\bar{z}(d, s, j-1), \bar{z}(d - \hat{w}_j, s - 1, j - 1) + p_j\}$$
$$\text{za } \ d = 0, \ldots, c;\ s = 1, \ldots, \Gamma \text{ in } j = 1, \ldots, n.$$

$$z(d, j) = max\{z(d, j-1), z(d - w_j, j - 1) + p_j\}$$
$$\text{za } \ d = 0, \ldots, c;\ j = \Gamma + 1, \ldots, n.$$

Začetna vrednost je $\bar{z}(d, s, 0) = -\infty$ za $d = 0, ..., c$ in s $= 0, ..., \Gamma$. Nato nastavimo $\bar{z}(0, 0, 0) = 0$. Oba zapisa sta med seboj povezana z enakostjo $z(d, \Gamma) = \bar{z}(d, \Gamma, \Gamma\}$ za vsak d. Optimalno vrednost robustnega problema nahrbtnika dobimo kot:

$$z^* = max \begin{cases} max\{z(d, n)|\ d = 1, ..., c \\ max\{\bar{z}(d, s, n)|\ d = 1, ..., c;\ s = 1, ..., \Gamma - 1\} \end{cases}$$

kjer porabimo tudi celotno kapaciteto nahrbtnika $c^* \leq c$.

Algoritem dinamičnega programiranja je sestavljen iz dveh korakov. V prvem koraku dobimo optimalno rešitev, ki vsebuje največ Γ elementov s povečano težo. V drugem koraku pa nato dobljeno rešitev lahko razširimo z dodatnimi elementi z nespremenjeno težo. Algoritem lahko razdelimo na dva koraka, ker razvrstitev predmetov po padajoči teži $\bar{w}_j$ zagotavlja, da so v vsaki rešitvi elementi z najmanjšimi indeksi (t.j. tisti, ki so bili v nahrbtnik položeni prej) tisti, ki bodo dosegli večjo težo.

4 NAČRT ZA NADALJEVANJE PROJEKTA

V nadaljevanju bova v programskem jeziku Python napisala program, ki s pomočjo dinamičnega programiranja poišče optimalni profit in pripadajočo optimalno množico predmetov. Nato bova program uporabila pri iskanju sestave portelja, kjer bova pokusila najti portfelj z najvišjim donosom v skladu z danimi omejitvami.