Laskuharjoitus 5

12.2.2003

1. *Latinalainen neliö* on $n \times n$-taulukko, jonka jokaisessa rivissä ja sarakkeessa kukin luvuista $1, \ldots, n$ esiintyy tasan kerran. Latinalaista neliötä kutsutaan redusoiduksi latinalaiseksi neliöksi, jos sen ensimmäisen rivin ja ensimmäisen sarakkeen alkiot esiintyvät järjestyksessä $1, 2, \ldots, n$. Kirjoita ohjelma, joka laskee peräytymishaulla redusoitujen $n \times n$ latinalaisten neliöiden lukumäärän.

2. Esitä jokin mielekäs kustannusfunktio ja naapuristo seuraaville vaikeille kombinatorisille optimointiongelmille.

   (a) Symmetrinen kauppamatkustajan ongelma.

   (b) Graafin väritys n värillä siten, että naapurisolmuilla on eri väri.

   (c) Graafin ositus: Jaa graafin solmut, joita on parillinen määrä, kahteen yhtäsuureen osaan siten, että osien välisten kaarien määrä on mahdollisimman pieni.

   (d) Lukujen ositus: jaa joukko $A = \{a_{1,}a_{2,} \ldots, a_n\}$ kahteen osaan A_1 ja A_2 siten, että $A_1 \cup A_2 = A$, $A_1 \cap A_2 = \emptyset$ ja

   $$\sum_{a \in A_1} a = \sum_{a \in A_2} a.$$

3. Esitä simuloitua jäähdytystä käyttävä algoritmi maksimiklikki ongelmalle.

Laskuharjoituksen 5 vastaukset

Tehtävä 1

Oheinen ohjelma toteuttaa tehtävän yksinkertaisella tavalla. Koska redusoidussa latinalaisessa neliössä ensimmäinen rivi ja sarake ovat määrätyt, $n \times n$-kokoisessa latinalaisessa neliössä riittää sijoittaa $(n-1)^2$ lukua, mikä siis on haun maksimisyvyys. Ohjelma ei pidä yllä konstruoitavaa latinalaista neliötä, vaan ainoastaan boolean-muuttujia `n_in_row_i` ja `n_in_col_j`, joista nähdään, joko luku n on laitettu riville i tai sarakkeeseen j.

```
/* Etsi kertalukua N olevat redusoidut latinalaiset neliöt */
#define N 4

int n_in_row_i[N+1][N+1]; /* onko numero n jo rivissä i     */
int n_in_col_j[N+1][N+1]; /* onko numero n jo sarakkeessa j */

int place(int depth) {
  int i,j,n,sum;

  if (depth==(N-1)*(N-1)) /* yksi neliö löytyi, koska on */
    return 1;             /* sijoitettu (N-1)**2 numeroa */

  i=2+depth%(N-1); /* lasketaan depthistä, mihin kohtaan */
  j=2+depth/(N-1); /* valitaan numero seuraavaksi       */

  sum=0;
  for (n=1; n<=N; n++) {
    if(!n_in_row_i[n][i] && !n_in_col_j[n][j]) {
      n_in_row_i[n][i]=1;  /* kullekin n:lle vuorollaan:         */
      n_in_col_j[n][j]=1;  /* jos n:ä ei ole laitettu riviin i   */
      sum+=place(depth+1); /* eikä sarakkeeseen j, laitetaan se  */
      n_in_row_i[n][i]=0;  /* näiden risteyskohtaan ja jatketaan */
      n_in_col_j[n][j]=0;  /* hakua; lopuksi otetaan luku pois   */
    }
  }
  return sum;
}

int main(int argc, char **argv) {
  int i,j;

  for(i=1;i<=N;i++) {      /* nollataan taulukot */
```

```
    for(j=1;j<=N;j++) {
      n_in_row_i[i][j]=n_in_col_j[i][j]=0;
    }
  }

  for(i=1;i<=N;i++) {   /* asetetaan luvut 1...n     */
    n_in_row_i[i][i]=1; /* ensimmäiseen sarakkeeseen */
    n_in_col_j[i][i]=1; /* ja riviin                 */
  }
  printf("%d\n", place(0));
  return 0;
}
```

Tehtävä 2 (a)

Kustannusfunktiona luonnollisin valinta on reitin pituus. Eräs mahdollisuus naapuristoksi on ns. 2-OPT -siirrot: muodostetaan nykyistä reittiä kuvaavasta graafista uusi reitti poistamalla kaksi kaarta ja lisäämällä kaksi kaarta.

Tehtävä 2 (b)

Tässä kuvataan ns. fixed-k lähestymistapa. Ratkaisu on mikä tahansa solmujen ositus k osaan (tyhjät osat sallitaan). Kaksi väritystä ovat toistensa naapurit, jos toinen saadaan toisesta siirtämällä jokin solmu osasta toiseen. Kustannusfunktiona esim. niiden kaarien lukumäärä, joiden molemmat solmut ovat samassa osituksen osassa. Naapuristoa voidaan rajoittaa seuraavasti: nykyistä ratkaisua voidaan muuttaa vain niiden solmujen osalta, joista lähtee ainakin yksi väritysehtoa rikkova kaari; tällöin tosin naapuristo ei enää ole symmetrinen.

Tehtävä 2 (c)

Yksinkertainen lähestymistapa: ratkaisujoukkona on kaikki solmujen ositukset kahteen osaan. Siirto on kahden solmun vaihto osien välillä. Kustannusfunktiona kaarien määrä osien välillä.

Parempi suorituskyky on saatu simuloidulla jäähdytyksellä, kun on sallittu mielivaltaiset ositukset kahteen osaan V_1 ja V_2. Siirtoina on nyt yhden solmun siirto osasta toiseen, ja kustannusfunktiossa on lisäksi sakkona summatermi

$$\alpha(|V_1| - |V_2|)^2,$$

joka lisää kustannusta aina kun osat ovat erikokoiset. Optimointialgoritmi saattaa sakkotermistä huolimatta päätyä epätasapainoiseen ratkaisuun, jolloin voidaan joko kasvattaa α:n arvoa tai lisätä jokin yksinkertainen heuristiikka, joka muutamaa solmua siirtämällä pääsee tasapainotettuun ratkaisuun.

Tehtävä 2 (d)

Otetaan ratkaisujoukoksi kaikki ositukset kahteen osaan. Määritellään k-naapuristo sellaiseksi, jossa yhdellä siirrolla saadaan siirtää enintään k alkiota osituksesta toiseen. Kustannusfunktioksi voidaan ottaa

$$\left| \sum_{a \in A_1} a - \sum_{a \in A_2} a \right| .$$

Tämä ongelma on hiukan hankala sikäli, että siinä on pahimmillaan hyvin paljon lokaaleja optimeja. Simuloitu jäähdytys esitetyllä naapuristolla ei pärjää erityisesti tähän ongelmaan kehitetylle Karmarkar-Karp -heuristiikalle, mutta parempaa on kehitetty.

Tehtävä 3

Maksimiklikin etsivä simuloitua jäähdytystä käyttävä algoritmi on esitetty pseudokoodilla seuraavasti:

```
Find_Max_Clique(G, c_max, T_0, α)
    c = 0
    T = T_0
    Choose feasible X ∈ 𝒳
    X_best = X
    while c ≤ c_max
        Y = h_N(X)
        if Y ≠ Fail
            if |Y| > |X|
                X = Y
                if |X| > |X_best|
                    X_best = X
            else
                r = random(0, 1)
                if r < e^((|Y|−|X|)/T)
                    X = Y
        c = c + 1
        T = αT
    return X_best
```

Jäähdytys tapahtuu lineaarisesti, ja α, c_{max} ja T_0 pitää valita sopivasti. Joukkona $\mathcal{X}$ on G:n aligraafien joukko ja $X \in \mathcal{X}$ on käypä, jos sen on klikki (eli kaikilla $x, y \in X, x \neq y$ on X:ssä kaari x:n ja y:n välillä). Kohdefunktiona on klikin (eli aligraafin) koko. X:n naapuristoksi $N(X)$ voidaan valita (esimerkiksi) G:n aligraafit, jotka saadaan lisäämällä tai poistamalla X:stä yksi solmu ja siihen liittyvät kaaret. Heuristiikkana h_N voidaan käyttää esimerkiksi valintaa “jokin käypä $Y \in N(X)$”.