T-79.5202 Kombinatoriset algoritmit

Harri Haanpää

Tietojenkäsittelyteorian laboratorio, TKK

Kevät 2006

T-79.5202 Kombinatoriset algoritmit

Combinatorial:

1: of, relating to, or involving combinations
2: of or relating to the arrangement of, operation on, and selection of discrete mathematical elements belonging to finite sets or making up geometric configurations

Algorithm:

a procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation; broadly : a step-by-step procedure for solving a problem or accomplishing some end especially by a computer

Kombinatorisia rakenteita

Lista: kokoelma alkioita tietyssä järjestyksessä, esim. $X = [0,1,3,0]$

Joukko: kokoelma eri alkioita, joita ei ole järjestetty, esim. $X = \{1,3,4\}$. $|X|$ on X:n alkioiden lkm. Karteesinen tulo $X \times Y = \{[x,y] \,|\, x \in X \wedge y \in Y\}$.

Osajoukko: Joukko X on joukon Y osajoukko, jos kaikille $x \in X$ myös $x \in Y$. Jos $|X| = k$, X on Y:n k-osajoukko.

Graafi: $G = (\mathcal{V}, \mathcal{E})$, missä $\mathcal{V}$ on solmujen ja $\mathcal{E}$ kaarien joukko. Kukin kaari on kahden solmun joukko.

Esimerkki: Latinalainen neliö

Joukkojärjestelmä: $(\mathcal{X}, \mathcal{B})$, missä $\mathcal{X}$ on perusjoukko ja $\mathcal{B}$ joukko $\mathcal{X}$:n osajoukkoja. (esim. $\mathcal{X}$:n ositus)

Latinalainen neliö: $n \times n$-taulukko, jonka joka rivissä ja sarakkeessa esiintyy kukin luvuista $\mathcal{Y} = \{1, \ldots, n\}$ tasan kerran, esim.

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{pmatrix}$$

joukkojärjestelmänä: $\mathcal{X} = \mathcal{Y} \times \{1, 2, 3\}$,
$\mathcal{B} = \left\{ \left\{ (y_1, 1), (y_2, 2), \left(A_{y_1 y_2}, 3\right) \right\} | y_1, y_2 \in \mathcal{Y} \right\}$

Transversaalisommitelma $TD_\lambda(k, n)$: $(\mathcal{X}, \mathcal{B})$, missä $|\mathcal{X}| = kn$; $\mathcal{X} = \mathcal{X}_1 \cup \ldots \cup \mathcal{X}_k$; $|\mathcal{X}_i| = n$; $|B \cap \mathcal{X}_i| = 1$ kaikilla $B \in \mathcal{B}$ ja $1 \leq i \leq 3$; kaikilla $x \in \mathcal{X}_i$, $y \in \mathcal{X}_j$, $i \neq j$ on λ lohkoa $B \in \mathcal{B}$, joille $\{x, y\} \subset B$.

Ongelmatyyppejä

- luettele tietynlaiset kombinatoriset rakenteet
  - luettele mahdolliset pokerikädet
- laske, montako tietynlaista rakennetta on
  - laske n bitin binäärisanat, joissa ei ole kahta peräkkäistä ykköstä
- etsi tietynlainen kombinatorinen rakenne
  - väritä graafin solmut 3 värillä siten, että naapurit väritetään eri väreillä

Hakuongelman variantteja

Selkäreppuongelma: Annetaan n esinettä, joiden painot ovat $w_1, \ldots w_n$ ja hyödyt $p_1, \ldots, p_n$. Reppuun voidaan pakata osajoukko $S \subseteq \{1, \ldots, n\}$, jos $\sum_{i \in S} w_i \leq M$, missä M on repun kapasiteetti. Tällöin saavutetaan hyöty $P(S) = \sum_{i \in S} p_i$.

1. Voidaanko reppuun pakata jokin esinevalikoima S, jolle $P(S) = P$?
   (NP-täydellinen päätösongelma!)
2. Konstruoi reppuun mahtuva esinevalikoima, jolle $P(S) = P$.
3. Mikä on suurin mahdollinen $P(S)$:n arvo?
4. Millä esinevalikoimalla saavutetaan suurin mahdollinen $P(S)$:n arvo?

Ratkaisustrategioita

Ahne algoritmi: rakennetaan ratkaisu valitsemalla joka askeleella "lyhytnäköisesti paras" vaihtoehto. Esim. selkäreppuongelmassa laitetaan reppuun tavaroita paino-hyöty-suhteen mukaisessa järjestyksessä kunnes reppu täyttyy.

Dynaaminen ohjelmointi: Kun optimiratkaisun osat ovat vastaavien osaongelmien optimaalisia ratkaisuja, ratkaistaan ensin osaongelmat ja yhdistellään niiden ratkaisut koko ongelman ratkaisuksi.

Hajoita ja hallitse: Jaetaan ongelma osaongelmiksi, ratkaistaan ne ja yhdistetään ratkaisut.

Peräytyvä haku: Käydään läpi koko kaikkien mahdollisten ratkaisujen vaihtoehtopuu.

Paikallinen haku: Koetetaan jotakin ratkaisua pienin askelin parantamalla löytää lopulta hyvä ratkaisu.

Tietorakenteita osajoukoille

Perusjoukon koko? Tarvittavat operaatiot? Alkion lisäys/poisto, jäsenyyden testaus, unioni, leikkaus, alkioiden lkm, alkioiden luetteleminen?

- (järjestetty) linkitetty lista alkioita
  - kun perusjoukko on suuri ja osajoukko pieni
- binääripuut
  - kun perusjoukko on suuri ja osajoukko pienehkö
- bittikarttaesitys
  - kun perusjoukko on pieni

esim. $S = \{1, 3, 11, 16\} \subset \{0, \ldots, 16\}$ voidaan esittää bittijonona 01010000000100001, joka voidaan pilkkoa esim. 8 bitin sanoiksi taulukkoon:
$A[0] = 01010000_2$, $A[1] = 00010000_2$,
$A[2] = 10000000_2$

Tietorakenteita graafeille ja joukkojärjestelmille

1. Kaarien joukko
2. Insidenssimatriisi: matriisi, jonka rivit vastaavat solmuja ja sarakkeet kaaria; alkio on 1, jos vastaava solmu on vastaavan kaaren päätepiste
3. Naapuruusmatriisi: matriisi, jonka rivit ja sarakkeet vastaavat solmuja; alkio on 1, jos vastaavien solmujen välillä on kaari
4. Naapuriluettelo: Annetaan kunkin solmun naapurien joukko

1. ja 2. soveltuvat myös joukkojärjestelmille.

rank- ja unrank-funktiot

Järjestetään lottorivit. Monesko lottorivi 3, 8, 12, 14, 15, 32, 38 on? Mikä lottorivi on rivi numero 3937483?

Olkoon S joukko joitakin kombinatorisia rakenteita. Numeroidaan ne $0 \ldots |S| - 1$:

$$\text{rank} : S \mapsto \{0, \ldots, |S| - 1\}$$

$$\text{unrank} : \{0, \ldots, |S| - 1\} \mapsto S$$

$$\text{rank}(s) = i \Leftrightarrow \text{unrank}(i) = s$$

- satunnainen $s = \text{unrank}(\text{random}(0 \ldots |S| - 1))$
- kokonaisluku on kompakti esitystapa

$$\text{successor}(s) = t \Leftrightarrow \text{rank}(t) = \text{rank}(s) + 1$$

$$\text{successor}(s) = \text{unrank}(\text{rank}(s) + 1),$$

kun $\text{rank}(s) < |S| - 1$

Rakenteet voidaan käydä läpi successor-funktiolla unrank (0):sta lähtien.

Listan leksikografinen järjestys

Määritellään järjestys listoille $l = [s_1, s_2, \ldots, s_n]$ ja $l' = [s'_1, s'_2, \ldots, s'_{n'}]$: Jos toinen lista on toisen alku, lyhempi edeltää pidempää. Muutoin etsitään pienin i, jolla $s_i \neq s'_i$. Jos $s_i \prec s'_i$, niin $l \prec l'$, ja kääntäen.

Esim. Tarkastellaan kolmen kirjaimen muodostamia listoja, merk. ['A','B','C'] ='ABC'.

Olkoon S = {'A','B','C',...,'Ö'}. Määritellään järjestys tavalliseen tapaan: $A \prec B$, jne.

rank_S('A') $= 0$, rank_S('N') $= 13$; $\text{unrank}_S(7)$ ='H'.

Nyt järjestys on 'AAA'$\prec$'AAB'$\prec \ldots \prec$'ÖÖÄ'$\prec$'ÖÖÖ', ja tässä erikoistapauksessa saadaan
$\text{rank}([s_1 s_2 s_3]) = |S|^2 \text{rank}(s_1) + |S| \text{rank}(s_2) + \text{rank}(s_3)$. (vrt. lukujärjestelmä)

Osajoukkojen leksikografinen järjestys

T	$\chi(T)$	rank (T)
$\emptyset$	$[0,0,0]$	0
$\{3\}$	$[0,0,1]$	1
$\{2\}$	$[0,1,0]$	2
$\{2,3\}$	$[0,1,1]$	3
$\{1\}$	$[1,0,0]$	4
$\{1,3\}$	$[1,0,1]$	5
$\{1,2\}$	$[1,1,0]$	6
$\{1,2,3\}$	$[1,1,1]$	7

Tutkitaan joukon $S = \{1,\ldots,n\}$ osajoukkoja.
Kun $T \subseteq S$, olkoon T:n karakteristinen vektori $\chi(T) = [x_{n-1},\ldots,x_0]$, missä $x_i = 1$, jos $n-i \in T$, ja $x_i = 0$, jos $n-i \notin T$. (vrt. bittikarttaesitys!)
Järjestetään osajoukot karakterististen vektorien mukaisesti leksikografiseen järjestykseen.

$$\text{rank}(T) = \sum_{i=0}^{n-1} x_i 2^i$$

Osajoukon leksikografinen rank-funktio

SubsetLexRank(n,T)
$r \leftarrow 0$
for $i \leftarrow 1$ to n
  if $i \in T$
    $r \leftarrow r + 2^{n-i}$
return r

Olkoon $V = \{1,\ldots,8\}$.
Esim. rank $(\{1,3,4,6\})$:

i	$i \in T$	2^{n-i}	r
1	true	128	128
2	false	64	128
3	true	32	160
4	true	16	176
5	false	8	176
6	true	4	180
7	false	2	180
8	false	1	180

Osajoukon leksikografinen unrank -funktio

SubsetLexUnrank(n,r)
$T \leftarrow \emptyset$
for $i \leftarrow n$ downto 1
  if r mod $2 = 1$
    $T \leftarrow T \cup \{i\}$
  $r \leftarrow \left\lfloor \frac{r}{2} \right\rfloor$
return T

Olkoon $V = \{1,\ldots,8\}$.
Esim. unrank (180):

i	r	mod 2	T
8	180	0	$\emptyset$
7	90	0	$\emptyset$
6	45	1	{6}
5	22	0	{6}
4	11	1	{4,6}
3	5	1	{3,4,6}
2	2	0	{3,4,6}
1	1	1	{1,3,4,6}

Jos perusjoukko ei ole $\{1,\ldots,n\}$ (esim. $\{0,\ldots,n-1\}$), voi kannattaa kuvata perusjoukko $\{1,\ldots,n\}$:lle.

Minimimuutosjärjestys

Toisinaan toivottavaa: kaksi peräkkäistä rakennetta eroavat mahdollisimman vähän. Osajoukoille etäisyys voi olla

$$\text{dist}(T_1,T_2) = |T_1 \Delta T_2|, \text{ missä}$$
$$T_1 \Delta T_2 = (T_1 \setminus T_2) \cup (T_2 \setminus T_1).$$

Esim. joukkojen subsetlexunrank $(n,3) = \{2,3\}$ ja subsetlexunrank $(n,4) = \{1\}$, etäisyys on 3, kun $n = 3$.

Osajoukoilla on olemassa järjestys, jossa peräkkäisten joukkojen dist on aina 1. Sellaisen järjestyksen karakteristiset vektorit muodostavat Gray-koodin.

Gray-koodit

Gray-koodi on 2^n n-bittisen binäärisanan lista, jossa peräkkäisten sanojen Hamming-etäisyys on 1.

Eräs mukava Gray-koodiperhe:
$G_1 = [0, 1]$, G_{i+1} saadaan G_i:stä ottamalla siitä kaksi kopiota, lisäämällä ensimmäisen kopion koodisanojen alkuun 0 ja toisen 1, kääntämällä jälkimmäisen kopion järjestys ja liittämällä kopiot yhteen:

$$G_2 = \begin{array}{|c|c|}\hline 0 & 0 \\ 0 & 1 \\ \hline 1 & 1 \\ 1 & 0 \\ \hline \end{array}$$

$$G_3 = \begin{array}{|c|c|}\hline 0 & 00 \\ 0 & 01 \\ 0 & 11 \\ 0 & 10 \\ \hline 1 & 10 \\ 1 & 11 \\ 1 & 01 \\ 1 & 00 \\ \hline \end{array}$$

GrayCodeSuccessor(n, T)
if $|T|$ is even
  return $T\Delta\{n\}$
else if max $(T) > 1$
  return $T\Delta\{\text{max (T)} - 1\}$
else
  return undefined

Olkoon koodisanan rank-luvun binääriesitys $b_{n-1} \ldots b_0$ ja vastaavan koodisanan $a_{n-1} \ldots a_0$.
$a_j = \left(b_j + b_{j+1}\right) \mod 2$ ja $b_j = \sum_{i=j}^{n-1} a_i \mod 2$.

k alkion osajoukkojen leksikografinen järjestys

$S = \{1, \ldots, n\}$. Generoidaan kaikki $\binom{n}{k}$ osajoukkoa, joissa on k alkiota.
Esitetään $T \subseteq S$ listana: $\vec{T} = [t_1, \ldots t_k]$, $t_i < t_{i+1}$, ja järjestetään osajoukot näiden listojen leksikografiseen järjestykseen.

T	$\vec{T}$	rank (T)
$\{1,2,3\}$	$[1,2,3]$	0
$\{1,2,4\}$	$[1,2,4]$	1
$\{1,2,5\}$	$[1,2,5]$	2
$\{1,3,4\}$	$[1,3,4]$	3
$\{1,3,5\}$	$[1,3,5]$	4
$\{1,4,5\}$	$[1,4,5]$	5
$\{2,3,4\}$	$[2,3,4]$	6
$\{2,3,5\}$	$[2,3,5]$	7
$\{2,4,5\}$	$[2,4,5]$	8
$\{3,4,5\}$	$[3,4,5]$	9

Seuraaja: kasvatetaan suurinta alkiota, jota voi kasvattaa, ja asetetaan sitä suuremmat alkiot minimiarvoihinsa.
rank $(T) = \sum_{i=1}^{k} \sum_{j=t_{i-1}+1}^{t_i-1} \binom{n-j}{k-i}$, missä $t_0 = 0$.

k alkion osajoukkojen co-lex-järjestys

T	$\overleftarrow{T}$	rank (T)
$\{1,2,3\}$	$[3,2,1]$	0
$\{1,2,4\}$	$[4,2,1]$	1
$\{1,3,4\}$	$[4,3,1]$	2
$\{2,3,4\}$	$[4,3,2]$	3
$\{1,2,5\}$	$[5,2,1]$	4
$\{1,3,5\}$	$[5,3,1]$	5
$\{2,3,5\}$	$[5,3,2]$	6
$\{1,4,5\}$	$[5,4,1]$	7
$\{2,4,5\}$	$[5,4,2]$	8
$\{3,4,5\}$	$[5,4,3]$	9

$S = \{1, \ldots, n\}$. Generoidaan kaikki $\binom{n}{k}$ osajoukkoa, joissa on k alkiota.
Esitetään $T \subseteq S$ listana: $\overleftarrow{T} = [t_1, \ldots t_k]$, $t_i > t_{i+1}$, ja järjestetään osajoukot näiden listojen leksikografiseen järjestykseen.
Seuraaja: kasvatetaan pienintä alkiota, jota voi kasvattaa; minimoidaan sitä pienemmät alkiot.
rank $(T) = \sum_{i=1}^{k} \binom{t_i-1}{k+1-i}$
rank on riippumaton n:stä!

Lex- ja co-lex-järjestysten yhteys

Kuvataan $T \subseteq \{1,\ldots n\}$ joukoksi $T' = \{n+1-t | t \in T\}$. T:n lex-järjestys on T':n käänteinen colex-järjestys, ja kääntäen!

T	T'	$\text{rank}_L(T)$	$\text{rank}_C(T')$
$\{1,2,3\}$	$\{5,4,3\}$	0	9
$\{1,2,4\}$	$\{5,4,2\}$	1	8
$\{1,2,5\}$	$\{5,4,1\}$	2	7
$\{1,3,4\}$	$\{5,3,2\}$	3	6
$\{1,3,5\}$	$\{5,3,1\}$	4	5
$\{1,4,5\}$	$\{5,2,1\}$	5	4
$\{2,3,4\}$	$\{4,3,2\}$	6	3
$\{2,3,5\}$	$\{4,3,1\}$	7	2
$\{2,4,5\}$	$\{4,2,1\}$	8	1
$\{3,4,5\}$	$\{3,2,1\}$	9	0

Tämän muunnoksen kautta on tehokkaampaa laskea lex-järjestyksen rank ja unrank.

Esimerkki k-osajoukon rank:sta

Järjestetään lottorivit leksikografiseen järjestykseen. Monesko lottorivi 3, 8, 12, 14, 15, 32, 38 on?

$$T = \{3,8,12,14,15,32,38\} \subseteq \{1,\ldots 39\}.$$
$$T' = \{37,32,28,26,25,8,2\}.$$

$$\begin{aligned}\text{rank}_C(T') &= \binom{37-1}{7} + \binom{32-1}{6} + \binom{28-1}{5} \\ &+ \binom{26-1}{4} + \binom{25-1}{3} + \binom{8-1}{2} + \binom{2-1}{1} \\ &= 9179387\end{aligned}$$

$$\begin{aligned}\text{rank}_L(T) &= \binom{39}{7} - 1 - \text{rank}_C(T') \\ &= 6201549\end{aligned}$$

Esimerkki k-osajoukon unrank:sta

Järjestetään lottorivit leksikografiseen järjestykseen. Mikä lottorivi on rivi numero 3937483?

$3937482 = \text{rank}_L(T) = \binom{39}{7} - 1 - \text{rank}_C(T')$

$T' = \text{unrank}_C\left(\binom{39}{7} - 1 - 3937482\right)$

i	r	t_i s.t. $\binom{t_i-1}{i} \le r < \binom{t_i}{i}$	$r - \binom{t_i-1}{i}$
7	11443454	38	1147982
6	1147982	33	40414
5	40414	24	6765
4	6765	22	780
3	780	18	100
2	100	15	9
1	9	10	0

$$T' = \{38,33,24,22,18,15,10\},$$
$$T = \{2,7,16,18,22,25,30\}$$

Permutaatiot

Permutaatio on tapa järjestää alkiot $\{1,\ldots,n\}$ eli bijektio joukolta $\{1,\ldots,n\}$ itselleen.

$$\pi : \{1,\ldots,n\} \mapsto \{1,\ldots,n\}$$

Esim.

x	1	2	3	4	5	6
$\pi(x)$	3	5	1	4	6	2

Permutaatio voidaan esittää listana: $[\pi(1), \pi(2), \ldots, \pi(n)]$

Esim. $[3,5,1,4,6,2]$.

Permutaatio voidaan esittää syklinotaatiossa, jossa suluissa aina edeltävä alkio kuvautuu seuraavalle ja viimeinen ensimmäiselle, esim.

$$\pi = (1,3)(2,5,6)(4) = (1,3)(2,5,6)$$

Permutaatioiden yhdistely

Permutaatiot ovat funktioita ja niitä yhdistellään kuin funktioita. Yhdistely tapahtuu siksi oikealta vasemmalle.

$$(\pi_1\pi_2)(x) = (\pi_1 \circ \pi_2)(x) = \pi_1(\pi_2(x))$$

$$\begin{aligned}(1,2)(2,3) &= (1,2,3)\\ (2,3)(1,2) &= (1,3,2)\end{aligned}$$

Permutaatioiden parillisuus

Yksinkertaisin permutaatio on transpositio (i,j), missä $i \neq j$. Permutaatiot voidaan jakaa kahteen luokkaan.

Parilliset permutaatiot voidaan ilmaista vain parillisen määrän transpositioita tulona, esim.

$$(1,2,3) = (1,2)(2,3)$$

Parittomat permutaatiot voidaan ilmaista vain parittoman määrän transpositioita tulona, esim.

$$(1,2,3,4) = (1,2)(2,3)(3,4)$$

Aputulos

Jos listat $d = [d_1, \ldots, d_n]$, missä $0 \leq d_i < n_i$ järjestetään leksikografiseen järjestykseen, niin

$$\text{rank}(d) = \sum_{i=1}^{n} d_i \prod_{j=i+1}^{n} n_j$$

unrank(r):
for $i = n$ downto 1:
  $d_i \leftarrow r \bmod n_i$
  $r \leftarrow \left\lfloor \frac{r}{n_i} \right\rfloor$

successor(d):
$i = n$
while $d_i = n_i - 1$
  $i \leftarrow i - 1$
$d_i \leftarrow d_i + 1$
for $j = i + 1$ to n
  $d_j = 0$

Permutaatioiden leksikografinen rank

Järjestetään permutaatiot listaesitystensä leksikografiseen järjestykseen. Esim.

$$[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]$$

Valittaessa listan i:nnettä alkiota on $n+1-i$ vaihtoehtoa. Merkitään d_i:llä, montako valittua pienempää vaihtoehtoa oli. Nyt $0 \leq d_i < n_i = n+1-i$, ja

$$\text{rank}(\pi) = \sum_{i=1}^{n-1} d_i (n-i)!$$

Esim. $\pi = [2,4,1,3] \Rightarrow d = [1,2,0,0]$ ja

$$\text{rank}(\pi) = 1 \cdot 3! + 2 \cdot 2! + 0 \cdot 1! = 10$$

Permutaatioiden leksikografinen unrank ja seuraaja

Vastaavasti unrank (10) antaa ensin $d = [1,2,0,0]$, ja siitä saadaan $\pi = [2,4,1,3]$.

Seuraaja: Pyritään olemaan koskematta alkupään alkioihin; etsitään listan lopusta lyhin osalista, joka ei ole käänteisessä lex. järjestyksessä. Vaihdetaan osalistan ensimmäinen alkio seuraavan suuremman alkion kanssa, ja käännetään osalistan loppu kasvavaan järjestykseen. Esim.

$\left[3,6,\underline{2,7,5,4,1}\right] \to [3,6,4,7,5,2,1] \to [3,6,4,1,2,5,7]$

Permutaatioiden minimimuutosjärjestys: Trotter–Johnson

Permutaatioiden minimimuutos on vierekkäisten alkioiden transpositio: $[\ldots,i,j,\ldots] \to [\ldots,j,i,\ldots]$.

Trotter (1962): lisätään alkioiden $\{1,\ldots,n-1\}$ minimimuutosjärjestykseen T^{n-1} alkio n seuraavasti. Kopioidaan kukin T^{n-1}:n alkio n-kertaisesti, ja lisätään sopiviin väleihin alkio n niin, että n:n paikat muodostavat siksak-kuvion.

$T^1 = [1]$, $T^2 = [[1,2],[2,1]]$

$$T^3 = \begin{bmatrix} & 1 & & 2 & 3 \\ & 1 & 3 & 2 & \\ 3 & 1 & & 2 & \\ 3 & 2 & & 1 & \\ & 2 & 3 & 1 & \\ & 2 & & 1 & 3 \end{bmatrix}$$

Menetelmä on tunnettu jo 1600-luvun Englannissa kirkonkellojen soitossa nimellä plain changes.

$$T^3 = \begin{bmatrix} [1,2,3] \\ [1,3,2] \\ [3,1,2] \\ [3,2,1] \\ [2,3,1] \\ [2,1,3] \end{bmatrix}$$

$$T^4 = \begin{bmatrix} & 1 & & 2 & & 3 & 4 \\ & 1 & & 2 & 4 & 3 & \\ & 1 & 4 & 2 & & 3 & \\ 4 & 1 & & 2 & & 3 & \\ 4 & 1 & & 3 & & 2 & \\ & 1 & 4 & 3 & & 2 & \\ & 1 & & 3 & 4 & 2 & \\ & 1 & & 3 & & 2 & 4 \\ & 3 & & 1 & & 2 & 4 \\ & 3 & & 1 & 4 & 2 & \\ & 3 & 4 & 1 & & 2 & \\ 4 & 3 & & 1 & & 2 & \\ 4 & 3 & & 2 & & 1 & \\ & 3 & 4 & 2 & & 1 & \\ & 3 & & 2 & 4 & 1 & \\ & 3 & & 2 & & 1 & 4 \\ & 2 & & 3 & & 1 & 4 \\ & 2 & & 3 & 4 & 1 & \\ & 2 & 4 & 3 & & 1 & \\ 4 & 2 & & 3 & & 1 & \\ 4 & 2 & & 1 & & 3 & \\ & 2 & 4 & 1 & & 3 & \\ & 2 & & 1 & 4 & 3 & \\ & 2 & & 1 & & 3 & 4 \end{bmatrix}$$

Trotter–Johnson rank

TJRank(π, n):
$\pi' \leftarrow \pi$ ilman alkiota n
$r \leftarrow$ TJRank $(\pi', n-1)$
jos r parillinen:
    $r \leftarrow nr +$ alkioiden lkm n : n oik. puolella
muuten:
    $r \leftarrow nr +$ alkioiden lkm n : n vas. puolella
return r

Esim. TJRank$([3,4,2,1],4)$:
    $r =$TJRank$([3,2,1],3)$
        $r =$TJRank$([2,1],2) = 1$
        r pariton; $r \leftarrow 3 \cdot 1 + 0 = 3$
    r pariton; $r \leftarrow 4 \cdot 3 + 1 = 13$

Trotter–Johnson unrank

TJUnrank(r, n):
$\pi' \leftarrow$ TJUnrank $\left(\left\lfloor \frac{r}{n} \right\rfloor, n-1\right)$
$r \leftarrow r$ mod n
jos π' parillinen:
    $\pi \leftarrow \pi'$, johon lisätään n s.e. sen oik. puolelle jää r alkiota
muuten:
    $\pi \leftarrow \pi'$, johon lisätään n s.e. sen vas. puolelle jää r alkiota
return π

Esim. TJUnrank$(13, 4)$:
    TJUnrank$(3, 3)$:
        TJUnrank$(1, 2) = [2, 1]$
    1 pariton: lisätään alkio 3 s.e. sen vas. puolelle jää
    3 mod $3 = 0$ alkiota: $[3, 2, 1]$
3 pariton: lisätään alkio 4 s.e. sen vas. puolelle jää 13 mod $4 = 1$ alkiota: $[3, 4, 2, 1]$

Trotter–Johnson successor

TJSuccessor(π, n):
$\pi' \leftarrow \pi$ ilman alkiota n
jos π' on parillinen ja n:ä voidaan siirtää vasempaan, tehdään niin
muuten jos π' on pariton ja n:ä voidaan siirtää oikealle, tehdään niin
muuten lasketaan TJSuccessor$(\pi', n-1)$ pitäen n:ä paikallaan

Esim. TJSuccessor$([4, 3, 1, 2])$:

$\pi' = [3, 1, 2]$ on parillinen, mutta 4:ä ei voi siirtää vasempaan; lasketaan $[4]$ +TJSuccessor$([3, 1, 2])$:

$\pi' = [1, 2]$ on parillinen, mutta 3:a ei voi siirtää vasempaan; lasketaan $[3]$ +TJSuccessor$([1, 2])$:

$\pi' = [1]$ on parillinen, ja 2:a voidaan siirtää vasempaan: $[2, 1]$

saadaan $[4] + [3] + [2, 1] = [4, 3, 2, 1]$

Myrvold & Ruskey: unrank

Sen sijaan, että valittaisiin jokin järjestys, jolle laskettaisiin sitten rank- ja unrank-funktioita, Myrvold ja Ruskey valitsivat nopean unrank-funktion ja kehittivät sitä vastaavan rank-funktion.

Perinteinen tapa luoda satunnainen permutaatio:

for $i = n$ downto 1
    swap$(\pi(i), \pi(r_i))$

missä $r_i =$ random $(1, \ldots, i)$. Syntyy permutaatio

$$\pi = (n, r_n)(n-1, r_{n-1}) \ldots (2, r_2)(1, r_1)$$

(tässä merkitään $(i, i) = (i)$ yksinkertaisuuden vuoksi). Jokaisella permutaatiolla on yksikäsitteinen esitystapa tässä muodossa.

Unrank on yksinkertainen: päätellään rank:sta r_i:ien arvot ja konstruoidaan permutaatio ylläolevalla algoritmilla.

Myrvold & Ruskey: rank

Rank:n laskemiseksi pitää ensin päätellä r_i:t ja sitten laskea

$$\text{rank}(\pi) = \sum_{i=1}^{n} (r_i - 1)(i-1)!$$

Koska π:n edellisen sivun esitysmuodossa alkiota n permutoidaan vain vasemmanpuoleisessa transpositiossa, $\pi(n) = r_n$. Näin π:stä voidaan helposti päätellä r_n. Sitten lasketaan $(n, r_n)\pi$, jossa n kuvautuu itselleen ja oleellisesti jää käsiteltäväksi joukon $\{1, \ldots, n-1\}$ permutaatio, ja iteroidaan.

Järjestys ei ole kovin intuitiivinen:

$0:2341$	$6:4123$	$12:3241$	$18:1423$
$1:4312$	$7:3124$	$13:3412$	$19:1324$
$2:2413$	$8:2431$	$14:4213$	$20:4231$
$3:2314$	$9:4132$	$15:3214$	$21:1432$
$4:3421$	$10:2143$	$16:4321$	$22:1243$
$5:3142$	$11:2134$	$17:1342$	$23:1234$

Kokonaislukujen ositus

$P(m)$: monellako tavalla positiivinen kokonaisluku m voidaan esittää positiivisten kokonaislukujen summana $m = a_1 + \ldots + a_n$, kun summattavien järjestyksellä ei ole merkitystä? (tai vastaavasti $a_1 \geq \ldots \geq a_n$)

$$P(5) = 7: \quad 5,\ 4+1,\ 3+2,\ 3+1+1,\ 2+2+1,\ 2+1+1+1,\ 1+1+1+1+1$$

$$P(1) = 1,\ P(2) = 2,\ P(3) = 3,\ P(4) = 5,\ P(5) = 7,\ P(6) = 11,$$

$$P(m) \sim \Theta\left(\frac{e^{\pi\sqrt{2m/3}}}{m}\right)$$

Ositusten generointi

GenRecPartition(m, B, L)
  if $m = 0$
    output L
  else
    for $i = 1$ to $\min(B, m)$:
      GenRecPartition$(m - i, i, L + [i])$

GenRecPartition$(m, m, [\,])$

Parametri m on ositettava kokonaisluku, parametri B on suurin kokonaisluku, joka voidaan laittaa seuraavaksi kiinnitettävään a_i:hin rikkomatta suuruusjärjestystä, ja L on lista osien pituuksista.

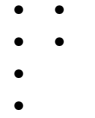
Ferrers-Young-kaaviot

Osituksen Ferrers-Young-kaavio saadaan kirjoittamalla kutakin a_i:tä vastaava määrä pisteitä omalle rivilleen.

$7 = 4 + 2 + 1 \Rightarrow D =$

Kääntämällä rivit sarakkeiksi saadaan vastaava konjugaattikaavio ja konjugaattiositus:

$D^* =$



$\Rightarrow 7 = 3 + 2 + 1 + 1$

$P(m, n)$:
m:n osituksia, joissa on n osaa, on yhtä monta kuin m:n osituksia, joissa suurin osa on n.

Relaatio I

Selvästi $P(m,m) = P(m,1) = 1$, kun $m > 1$. Määritellään $P(m,0) = 0$, kun $m > 0$, ja $P(0,0) = 1$.

Teoreema 3.2: Kun $m \geq n > 0$,

$$P(m,n) = P(m-1,n-1) + P(m-n,n).$$

Tod. Merkitään $\mathcal{P}(m,n)$:llä m:n n-ositusten joukkoa. Jaetaan $\mathcal{P}(m,n)$ kahtia ja määritellään kuvaukset:

jos $a_n = 1$, $\Phi_1([a_1,\ldots,a_n]) = [a_1,\ldots,a_{n-1}]$;
jos $a_n > 1$, $\Phi_2([a_1,\ldots,a_n]) = [a_1-1,\ldots,a_n-1]$

Φ_1 ja Φ_2 ovat bijektioita $\mathcal{P}(m,n)$:n osilta joukoille $\mathcal{P}(m-1,n-1)$ ja $\mathcal{P}(m-n,n)$, joten joukkojen alkioiden lukumäärät ovat identtiset.

Relaatioita II

Teoreema 3.3: Kun $m \geq n > 0$,

$$P(m,n) = \sum_{i=0}^{n} P(m-n,i)$$

Tod: Jaetaan $\mathcal{P}(m,n)$ osiin $\mathcal{P}(m,n)_i$, joista kuhunkin kuuluvat ositukset, joissa on tasan i osaa, jotka ovat suurempia kuin 1.

Kullekin $0 \leq i \leq n$ määritellään bijektio

$$\Phi_i : \mathcal{P}(m,n)_i \mapsto \mathcal{P}(m-n,i)$$

seuraavasti:

$$\Phi_i([a_1,\ldots,a_n]) = [a_1-1,\ldots a_i-1]$$

Rank-funktio $\mathcal{P}(m,n)$:lle

Järjestetään ositukset $[a_1,\ldots,a_n]$ käänteisen standardimuodon $[a_n,\ldots,a_1]$ mukaiseen leksikografiseen järjestykseen. Esim. $\mathcal{P}(10,4)$:

std. muoto	käänt. std. muoto
$[7,1,1,1]$	$[1,1,1,7]$
$[6,2,1,1]$	$[1,1,2,6]$
$[5,3,1,1]$	$[1,1,3,5]$
$[4,4,1,1]$	$[1,1,4,4]$
$[5,2,2,1]$	$[1,2,2,5]$
$[4,3,2,1]$	$[1,2,3,4]$
$[3,3,3,1]$	$[1,3,3,3]$
$[4,2,2,2]$	$[2,2,2,4]$
$[3,3,2,2]$	$[2,2,3,3]$

Ositukset, joissa $a_n = 1$, tulevat tässä järjestyksessä ennen niitä, joissa $a_n > 1$. Saadaan

$$\text{rank}([a_1,\ldots,a_n]) = \begin{cases} \text{rank}([a_1,\ldots,a_{n-1}]) & \text{jos } a_n = 1 \\ \text{rank}([a_1-1,\ldots,a_n-1]) + P(m-1,n-1) & \text{jos } a_n > 1 \end{cases}$$

Seuraajafunktio $\mathcal{P}(m,n)$:ssä

Kirjoitetaan tässä poikkeuksellisesti listat käänteisen standardimuodon mukaisesti alkioidensa kasvavaan järjestykseen, $a_i \leq a_{i+1}$.

Partitio $[a_1,\ldots,a_n]$ on viimeinen, kun $a_1 + 1 \geq a_n$. Tällöin m on jaettu mahdollisimman tasan n:llä.

Leksikografisessa järjestyksessä koetetaan seuraajaa hakiessa pitää listan alkupään alkiot muuttumattomina.

Seuraajafunktio:

1. etsitään listan viimeinen osalista, joka ei ole tasan jaettu, ts. suurin i, jolle $a_i + 1 < a_n$.
2. Kasvatetaan a_i:tä yhdellä ja asetetaan $a_{i+1},\ldots,a_{n-1}$ minimiarvoonsa ($= a_i$)
3. Täsmätään summa asettamalla $a_n = m - \sum_{i=1}^{n-1} a_i$.

Esim.:
$[1,2,4,5,5]$:lle $i = 2$. Asetetaan $a_2 = a_2 + 1 = 3$, $a_3 = 3$, $a_4 = 3$ ja $a_5 = 17 - 3 - 3 - 3 - 1 = 7$; saadaan $[1,3,3,3,7]$.

Nimetyt puut

Graafi $G = (V, E)$ on puu, kun se on yhtenäinen ja syklitön.
Solmun v asteluku on niiden kaarien määrä, joissa solmu esiintyy.
Olkoon $V = \{1, 2, \ldots, n\}$. Tällöin on n^{n-2} eri puuta, joiden solmujoukko on V.

Olkoon $\mathcal{T}_n$ niiden puiden joukko, joiden solmujoukko on V.
Prüferin vastaavuus:

$$\text{Prüfer} : \mathcal{T}_n \mapsto V^{n-2}$$

$$\text{Prüfer}^{-1} : V^{n-2} \mapsto \mathcal{T}_n$$

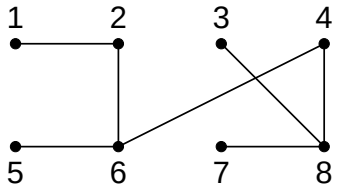
Prüfer

Prüfer$(n, \mathcal{E})$:
for $i = 1$ to $n - 2$:
    olkoon v korkeanumeroisin solmu, jonka asteluku=1
    etsitään kaari $\{v, v'\} \in \mathcal{E}$ ja asetetaan $L_i \leftarrow v'$
    poistetaan graafista kaari $\{v, v'\}$

Kukin solmu v esiintyy listassa L deg $(v) - 1$ kertaa. Graafiin jää kaari $\{1, v\}$, missä v:n asteluku on 1 algoritmin ajamisen jälkeen.

InvPrüfer(n, L):
lasketaan listasta L solmujen asteluvut
lisätään listan jatkoksi ykkönen: $L_{n-1} \leftarrow 1$
for $i = 1$ to $n - 1$:
    olkoon v korkeanumeroisin solmu, jonka asteluku=1
    lisätään graafiin kaari $\{v, L_i\}$
    vähennetään v:n ja L_i:n astelukua yhdellä

Prüfer-esimerkki

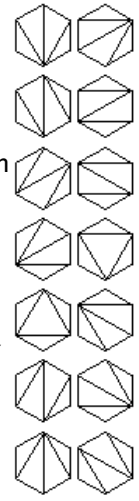


Solmujen asteluvut	L_i	Kaari
[1,2,1,2,1,3,1,3]	8	{7,8}
[1,2,1,2,1,3,0,2]	6	{5,6}
[1,2,1,2,0,2,0,2]	8	{3,8}
[1,2,0,2,0,2,0,1]	4	{4,8}
[1,2,0,1,0,2,0,0]	6	{4,6}
[1,2,0,0,0,1,0,0]	2	{2,6}
[1,1,0,0,0,0,0,0]		

Listaesityksestä saadaan helposti rank- ja unrank-funktiot tulkitsemalla lista n-kantaisena lukuna.

Catalanin luvut

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

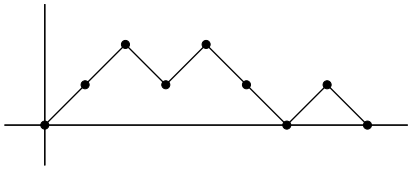


Catalanin luvut esiintyvät monessa yhteydessä:

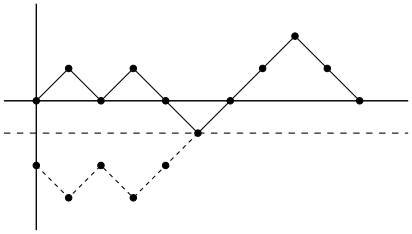
- monellako eri tavalla matriisitulolauseke voidaan suluttaa niin, että aina kaksi tekijää kerrotaan kerrallaan: $((M_1\,(M_2M_3))\,(M_4M_5))$
- monellako eri tavalla konveksi $n + 2$-kulmio voidaan kolmioida
- montako sellaista $2n$ bitin jonoa on, joissa on n ykköstä ja n nollaa, ja joissa minkään kohdan vasemmalla puolella ei ole enemmän ykkösiä kuin nollia: 000111, 001011, 001101, 010011, 010101

Catalanin luvuista

Em. kaltaiset binääriluvut voidaan esittää vuoristona, joka ei laske 0-tason alle. Esim. $a = 00101101$ vastaa vuoristoa



Peilataan ne vuoristot, jotka laskevat 0-tason alle, akselin $y = -1$ yli alusta siihen asti, jossa ne ensimmäisen kerran laskevat 0-tason alle. Saadaan bijektio 0-tason alle laskevien ja $(-2, 0)$:sta $(2n, 0)$:aan kulkevien vuoristojen välille.



$$C_n = \binom{2n}{n} - \binom{2n}{n+1} = \left(1 - \frac{n}{n+1}\right)\binom{2n}{n} = \frac{1}{n+1}\binom{2n}{n}$$

Catalan-rank ja -unrank

Lasketaan, monellako tavalla vuoriston voi saattaa loppuun kustakin kohdasta, esim. tapaus C_5:

5						1					
4					5		1				
3				14		4		1			
2			28		9		3		1		
1		42		14		5		2		1	
0	42		14		5		2		1		1
	0	1	2	3	4	5	6	7	8	9	10

rank: seurataan vuoristoa; kun se menee oikealle alas, lisätään rank:iin luku, joka olisi ollut oikealla ylhäällä.

unrank: jos rank $\geq$ oikealla ylhäällä oleva luku, mennään oikealle alas ja vähennetään oikealla ylhäällä ollut luku rank:sta; muuten mennään oikealle ylös

Esim. rank $(0010110101) = 22$

Peräytyvä haku

- yleinen menetelmä kombinatorisen haku-, optimointi-, generointi- tai enumerointiongelman ratkaisemiseen.
- rekursiivinen: toteutetaan tyypillisesti itseään kutsuvilla aliohjelmilla, jotka rakentavat ratkaisun askel askeleelta
- täydellinen haku: koko ratkaisuavaruus käydään läpi
- karsinta säästää tarkastelemasta epäoleellisia hakuavaruuden osia

Esimerkki peräytyvästä hausta

Selkäreppuongelma: Annetaan n esinettä, joiden painot ovat $w_1, \ldots w_n$ ja hyödyt $p_1, \ldots, p_n$. Repun kapasiteetti on M. Maksimoidaan $P(x) = \sum p_i x_i$ rajoitusehdoilla $x_i \in \{0, 1\}$ ja $\sum w_i x_i \leq M$.

Konstruoidaan rekursiivisesti lista $[x_0, \ldots, x_{n-1}]$. Tässä $\text{len}(x)$ on listan x pituus eli jo kiinnitettyjen muuttujien x_i lukumäärä; rekursio käynnistetään kutsulla Knapsack1$([])$.

```
Knapsack1(x):
if len(x) = n:
    if ∑_i w_i x_i ≤ M:
        CurP ← ∑_i p_i x_i
        if CurP > OptP:
            OptP ← CurP
            OptX ← x
else:
    Knapsack1(x + [1])
    Knapsack1(x + [0])
```

Peräytyvä haku yleisesti

Monen kombinatorisen ongelman ratkaisut voidaan esittää listana $X = [x_0, \ldots, x_{n-1}]$, missä $x_i \in P_i$, missä P_i on äärellinen x_i:n mahdollisten arvojen joukko. Peräytyvä haku konstruoi joukon $P_0 \times P_1 \times \ldots \times P_{n-1}$ kaikki alkiot. Haun aikana konstruoitavan listan pituus vastaa hakusolmun syvyyttä hakupuussa.

Osittainen ratkaisu $[x_0, \ldots, x_{l-1}]$ voi rajoittaa hakua; joskus voidaan päätellä, etteivät jotkin $x_l \in P_l$ voi johtaa käypiin ratkaisuihin. Tällöin voidaan karsia hakupuuta ja rajoittaa haku vaihtoehtojoukkoon $C_l \subseteq P_l$.

Backtrack(x):
jos $x = [x_0, \ldots, x_{l-1}]$ on käypä ratkaisu:
    käsittele se
laske C_l
kullekin $c \in C_l$:
    Backtrack($x + [c]$)

Knapsack2(x):
if len(x) $= n$:
    if $CurP > OptP$:
        $OptP \leftarrow CurP$
        $OptX \leftarrow x$
if len(x) $= n$:
    $C_l \leftarrow \emptyset$
else if $\sum_{i=0}^{l-1} w_i x_i + w_l \leq M$:
    $C_l \leftarrow \{0, 1\}$
else:
    $C_l \leftarrow \{0\}$
for $c \in C_l$:
    Knapsack2($x + [c]$)

Klikkien generointi

Klikki on graafin $G = (\mathcal{V}, \mathcal{E})$ solmujoukon $\mathcal{V}$ sellainen osajoukko $S \subseteq \mathcal{V}$, että kaikkien S:n solmuparien $x, y \in S$, $x \neq y$ välillä on kaari: $\{x, y\} \in \mathcal{E}$.

Maksimaalinen klikki on klikki, joka ei ole suuremman klikin osajoukko.

Määritellään peräytyvä haku:
$[x_0, \ldots, x_{l-1}]$ vastaa klikkiä $S_l = \{x_0, \ldots, x_{l-1}\}$

$$\begin{aligned} C_l &= \{v \in \mathcal{V} \setminus S_{l-1} : \{v, x\} \in \mathcal{E} \text{ kaikilla } x \in S_{l-1}\} \\ &= \{v \in C_{l-1} \setminus \{x_{l-1}\} : \{v, x_{l-1}\} \in \mathcal{E}\} \end{aligned}$$

Ongelma: algoritmi generoi kaikki k solmun klikit $k!$ kertaa, kerran kussakin järjestyksessä! Ratkaisu: määritellään solmuille järjestys $v_0 < \ldots < v_{n-1}$ ja valitaan

$$C_l = \{v \in C_{l-1} : \{v, x_{l-1}\} \in \mathcal{E} \wedge v > x_{l-1}\}$$

Klikkien generointi II

Esilasketaan aluksi kullekin solmulle v apujoukot $N_v = \{u \in \mathcal{V} : \{u, v\} \in E\}$ ja $G_v = \{u \in \mathcal{V} : u > v\}$. N_v on solmun v naapurien joukko ja G_v on valitussa järjestyksessä solmun v jälkeen tulevien solmujen joukko.

Haun aikana X on lista solmuja, jotka muodostavat klikin; N on X:n solmujen yhteisten naapurien joukko; ja C on yhteisten naapurien joukko, jotka ovat järjestyksessä viimeisen X:ään lisätyn solmun jälkeen.

AllCliques(X, N, C):
output X
if $N = \emptyset$:
    X on maksimaalinen
for $v \in C$:
    AllCliques($X + [v], N \cap N_v, C \cap N_v \cap G_v$)

AllCliques($[], \mathcal{V}, \mathcal{V}$)

Hakupuun koon arviointi

Jos hakupuun joka kohdassa $|C_i| = c_i$, puun koko on $|T| = 1 + c_0 + c_0c_1 + c_0c_1c_2 + \ldots + c_0c_1 \ldots c_{n-1}$. Yleensä näin ei ole. Kutsutaan hakupuun solmuja nimillä $[x_0, \ldots x_{l-1}]$ sen mukaan, mitä valintoja tekemällä niihin päästään. Puun kokoa voidaan arvioida valitsemalla joka askeleella tasajakautuneesti satunnainen vaihtoehto, jolloin todennäköisyys, että polku kulkee solmun X kautta, on

$$p(X) = \begin{cases} 1 & \text{kun } l = 0 \\ \frac{p(f(X))}{|C_{l-1}(f(X))|} & \text{kun } l > 0, \end{cases}$$

missä $f([x_0, \ldots x_{l-1}])=[x_0, \ldots, x_{l-2}]$ (solmun isä). Merkitään $m(X) = 1$, jos X kuuluu polkuun, ja $m(X) = 0$, jos ei. Arvioidaan puun kokoa laskemalla

$$N = \sum_{X \in P} \frac{1}{p(X)} = \sum_{X \in T} \frac{m(X)}{p(X)}.$$

Väite: $E(N) = |T|$. Todistus:

$$E(N) = E \sum_{X \in T} \frac{m(X)}{p(X)} = \sum_{X \in T} \frac{E(m(X))}{p(X)}$$

$$= \sum_{X \in T} \frac{p(X)}{p(X)} = \sum_{X \in T} 1 = |T|.$$

Esimerkki: Sudoku

	8				4	3		
	6		2		8	5		4
					7			2
				2	1	4		
	7	1	9		5	6	3	
		4	8	3				
8			5					
1		9	4		2		6	
		7	2				8	

Sudokussa annetaan osittain täytetty $n \times n$-ruudukko, joka on jaettu $p \times q$-osaruudukkoihin. Ruudukko tulee täydentää latinalaiseksi neliöksi: jokaisen numeroista $1 \ldots n$ tulee esiintyä kussakin rivissä ja kussakin sarakkeessa kerran. Lisäksi jokaisen numeroista tulee esiintyä kerran kussakin osaruudukossa. (Yleensä $p = q = 3$ ja $n = 9$.)

Suoraviivaisin tapa soveltaa peräytyvää hakua olisi tarkastella ruudukkoa ruutu ruudulta ja valita aina ruutuun jokin numero, joka siihen muissa ruuduissa jo olevien numeroiden puolesta sopii.

Esimerkki: Sudoku

Peräytyvä haku tulee usein tehokkaammaksi, kun valitaan seuraavaksi kiinnitettäväksi muuttujaksi se, joka voidaan kiinnittää pienimmällä määrällä tapoja. Esimerkiksi sudokussa voidaan valita seuraavaksi täydennettäväksi ruutu, johon käy pienin määrä vaihtoehtoja.

Jos vaihtoehtoja on 0, ratkaisua ei löydy; jos vaihtoehtoja on 1, saatiin pääteltyä ratkaisusta lisää haarautumatta, ja tämä auttaa jatkossakin rajoittamaan haarautumisvaihtoehtojen lukumäärää.

Sudoku voitaisiin muuten esittää maksimiklikkiongelmanakin: solmujoukon muodostavat kaikkiaan n^3 rivi-sarake-arvoyhdistelmää, ja kahden solmun välillä on kaari, jos ne ovat yhteensopivia (eli eivät esimerkiksi sisällä samaa arvoa samassa sarakkeessa). Jos tästä verkosta nyt löytyy n^2 solmun klikki, se vastaa täydennettyä ruudukkoa.

Täsmällinen peite (Exact Cover)

Kun annetaan joukko R ja joukko S sen osajoukkoja, voidaanko valita osajoukkojen osajoukko, joka osittaa R:n?

$R = \{0, \ldots, n-1\}$. $S = \{S_0, \ldots, S_{m-1}\}$, missä $S_i \subseteq R$ kaikilla i. Onko olemassa $S' \subseteq S$, jolle $\bigcup_{X \in S'} X = S$ ja $S_i \cap S_j = \emptyset$, kun $S_i,\ S_j \in S'$?

Graafin $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, missä $\mathcal{V} = \{0, \ldots, m-1\}$ ja $\mathcal{E} = \left\{\{i, j\} : S_i \cap S_j = \emptyset\right\}$, klikit vastaavat ongelman osittaisratkaisuja. Voitaisiin suoraan soveltaa AllCliques-algoritmia ja tarkastaa, onko jokin löydetyistä maksimaalisista klikeistä ratkaisu.

Täsmällinen peite II

AllCliques-algoritmissa valitaan solmuille järjestys. Valitaan osajoukoille laskeva leksikografinen järjestys. Merkitään H_i:llä niiden osajoukkojen joukkoa, joiden pienin alkio on i. H_i:n joukot edeltävät H_{i+1}:n joukkoja.

AllCliques-algoritmissa C_l muodostuu solmuista, jotka ovat järjestyksessä listassa jo olevien jälkeen ja kaikkien listassa olevien solmujen naapureita. (tässä: eivät sisällä yhteisiä alkioita ko. osajoukkojen kanssa)

Täsmällinen peite -ongelmassa, koska kuitenkin jokainen perusjoukon alkio tulee peittää, voimme lisätä rakenteilla olevaan peitteeseen aina jonkin joukon, joka peittää pienimmän alkion, jota ei ole vielä peitetty: jos r on pienin alkio, jota jo valitut joukot eivät peitä, valintajoukko $C'_l = C_l \cap H_r$ riittää.

(Tässäkin voisi olla tehokkaampaa peittää seuraavaksi aina se alkio, jonka peittämiseksi on vähiten vaihtoehtoja.)

Sudoku täsmällinen peite -ongelmana

Sudoku voidaan varsin suoraviivaisesti nähdä täsmällinen peite -ongelmana. Jokaisen rivi-arvo-, sarake-arvo, osaruudukko-arvo- ja rivi-sarakeyhdistelmän tulee esiintyä kerran. Jos rivien, sarakkeiden, osaruudukkojen ja arvojen joukot ovat vastaavasti $R = \{r_1, \ldots, r_n\}$, $C = \{c_1, \ldots, c_n\}$, $B = \{b_1, \ldots, b_n\}$ ja $V = \{v_1, \ldots, v_n\}$, niin peitettävä joukko on

$$(R \times V) \cup (C \times V) \cup (B \times V) \cup (R \times C).$$

Jos nyt kirjoitamme riville r_i, sarakkeeseen c_j ja osaruudukkoon b_k arvon v_l, tulemme peittäneeksi joukon

$$\{(r_i, v_l), (c_j, v_l), (b_k, v_l), (r_i, c_j)\}.$$

Tällaisia joukkoja on kaikkiaan n^3, ja peitteeseen tulee niistä n^2.

Rajoitusfunktiot

Optimointitehtävässä hakupuuta voidaan karsia arvioimalla, miten hyviä ratkaisuja jostakin haarasta voi löytyä.

Olkoon profit (X) ratkaisun X hyöty. Olkoon $P(X)$ suurin hyöty, joka voidaan saavuttaa osittaisratkaisun X jälkeläisissä. Olkoon $B(X)$ helposti laskettava funktio, jolla arvioidaan $P(X)$:ää s.e. $B(X) \geq P(X)$.

Jos paras aiemmin löydetty ratkaisu on X', käsitellään osittaisratkaisua X, ja profit $(X') > B(X)$, voidaan haara karsia, sillä profit$(X') > B(X) \geq P(X)$, eikä X:n jälkeläisten joukossa voi olla X':a parempaa ratkaisua.

```
Bounding(X):
jos X on käypä ratkaisu:
        P ← profit(X)
                if P > OptP :
                        OptP ← P
                        OptX ← X
laske C_l
B ← B(X)
kullekin x ∈ C_l:
        if B ≤ OptP: # jos karsitaan myös
                return # yhtäsuuruudella, testin on
                        # oltava tässä, sillä
                        # OptP voi muuttua
        Bounding(X + [x])
```

Rationaaliselkäreppu

Eräs tapa muodostaa rajoitusfunktio on höllentää joitakin alkuperäisen tehtävän rajoituksista siten, että ratkaiseminen helpottuu.

Selkäreppuongelma: Annetaan n esinettä, joiden painot ovat $w_1, \ldots w_n$ ja hyödyt $p_1, \ldots, p_n$. Repun kapasiteetti on M. Maksimoidaan $P(x) = \sum p_i x_i$ rajoitusehdoilla $x_i \in \{0, 1\}$ ja $\sum w_i x_i \leq M$.

Rationaaliselkäreppuongelma: kuten edellä, mutta vaaditaan $x_i \in \{0, 1\}$:n sijaan, että $0 \leq x_i \leq 1$.

Rationaaliselkäreppu

Annetaan n esinettä, joiden painot ovat $w_1, \ldots w_n$ ja hyödyt $p_1, \ldots, p_n$. Repun kapasiteetti on M. Maksimoidaan $P(x) = \sum p_i x_i$ rajoitusehdoilla $0 \leq x_i \leq 1$ ja $\sum w_i x_i \leq M$.

Kokonaislukuarvoisilla p_i, w_i, M muuttujat x_i tulevat olemaan rationaalisia. Ahne algoritmi antaa optimin:

RKnap$(p_1, \ldots, p_n, w_1, \ldots, w_n, M)$
järjestä esineet s.e. $p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n$
for $i = 1$ to n:
    $x_i \leftarrow \min\left(1, \frac{M - \sum_{j=1}^{i-1} w_i x_i}{w_i}\right)$
return $\sum p_i x_i$

Selkäreppu

Knapsack3$(x, CurW)$:
if len$(x) = n$:
    if $\sum_i p_i x_i > OptP$:
        $OptP \leftarrow \sum_i p_i x_i$
        $OptX \leftarrow X$
if $l = n$:
    $C_l \leftarrow \emptyset$
else if $CurW + w_{l+1} \leq M$:
    $C_l \leftarrow \{0, 1\}$
else:
    $C_l \leftarrow \{0\}$
$B \leftarrow \sum_{i=1}^{l} p_i x_i +$ RKnap$(p_{l+1}, \ldots, p_n, w_{l+1}, \ldots w_n, M - CurW)$
for $c \in C_l$:
    if $B \leq OptP$
        return
    Knapsack3$(x + [c], CurW + w_{l+1} x_{l+1})$

Tämä algoritmi olettaa, että $\frac{p_1}{w_1} \geq \ldots \geq \frac{p_n}{w_n}$.
Kirjan testeissä tietynlaisilla satunnaisilla ongelmilla tämä karsinta pienensi hakupuuta dramaattisesti (parhaimmillaan muttei epätyypillisesti 18953093→180).

Kauppamatkustajan ongelma

Annetaan $K_n = (\mathcal{V}, \mathcal{E})$, n solmun täydellinen *suunnattu* graafi, ja kustannusfunktio cost : $\mathcal{E} \mapsto \mathbb{Z}^+$. Etsi Hamiltonin kierros X, jolle cost$(X) = \sum_{e \in E(X)}$ cost(e) on pienin. (Hamiltonin kierros on polku, joka käy kaikissa solmuissa ja palaa lähtöpisteeseensä.)
Hamiltonin kierros voidaan esittää solmujen permutaationa, ja kierros voidaan valita alkamaan solmusta 1. Kierros 3 6 2 1 4 5 7 3 voidaan siten esittää listana $[1, 4, 5, 7, 3, 6, 2]$.

TSP1(x):
if len$(x) = n$:
    $C \leftarrow$ cost(X)
    if $C < OptC$:
        $OptC \leftarrow C$
        $OptX \leftarrow X$

if len$(x) = 0$:
    $C_l \leftarrow \{1\}$
else if len$(x) = 1$:
    $C_l \leftarrow \{2, \ldots n\}$
else
    $C_l \leftarrow C_{l-1} \setminus \{x_{l-1}\}$

for each $c \in C_l$:
    TSP1$(x + [c])$

Rajoitusfunktioita kauppamatkustajalle

cost-funktio voidaan esittää matriisina M, jossa m_{ij} on (suunnatun!) kaaren (i,j)kustannus.

$$\mathrm{M} = \begin{bmatrix} \infty & 3 & 5 & 8 \\ 3 & \infty & 2 & 7 \\ 5 & 2 & \infty & 6 \\ 8 & 7 & 6 & \infty \end{bmatrix}$$

MinEdgeBound: Lasketaan kunkin sarakkeen (rivin) pienimmät arvot yhteen; pitäähän jokaiseen solmuun tulla jostakin (jokaisesta mennä jonnekin).

ReduceBound: Jos jonkin rivin (sarakkeen) kaikista alkioista vähennetään k, kierroksen pituus pienenee k:lla. Siis: olkoon c sarakkeiden pienimpien alkioiden summa. Vähennetään kunkin sarakkeen kaikista alkioista pienin alkio. Lasketaan saadusta matriisista vastaavasti riveittäin r. Näin joka riville ja sarakkeeseen tulee ainakin yksi 0. Alaraja: $c + r$

Rajoituksia kauppamatkustajalle II

Osittaisratkaisua $X = [x_0, \ldots, x_{l-1}]$ vastaava alaraja saadaan seuraavasti: käsitellään X:ä kuin se olisi yksi solmu, josta solmuun y siirtyminen aiheuttaa kustannuksen cost $((x_{l-1}, y))$ ja johon siirtyminen solmusta aiheuttaa kustannuksen cost $((y, x_0))$.

Poistetaan kustannusmatriisista rivit $x_0, \ldots, x_{l-2}$ ja sarakkeet $x_1, \ldots, x_{l-1}$ sekä asetetaan $m_{l-1,0} = \infty$.

Esimerkiksi osittaisratkaisulla $[1, 2]$ saadaan

$$\mathrm{M} = \begin{bmatrix} \infty & 3 & 5 & 8 \\ 3 & \infty & 2 & 7 \\ 5 & 2 & \infty & 6 \\ 8 & 7 & 6 & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & 2 & 7 \\ 5 & \infty & 6 \\ 8 & 6 & \infty \end{bmatrix}$$

Näin ongelma on saatu redusoitua $n - l + 1$ solmun suunnatuksi kauppamatkustajan ongelmaksi, johon voidaan soveltaa edellisiä alarajoja.

Rajoituksia maksimiklikkiongelmalle

AllCliques-proseduurissa C_l oli niiden solmujen joukko, jotka ovat osittaisratkaisun yhteisiä naapureita, ja tulevat järjestyksessä osittaisratkaisun solmujen jälkeen.

Rajoitus: $B(X) = |X| + |C_l|$

Rajoja graafinväritysongelmasta: Jos graafin solmut voidaan värittää k värillä ilman, että yhdenkään kaaren päätepisteet ovat samanvärisiä, sen suurin klikki on kooltaan korkeintaan k (klikin kaikki solmut ovat naapureita ja siten erivärisiä).

Rajoitus: väritetään solmujen C_l indusoima graafi. Jos tämä onnistuu k värillä, $B(X) = |X| + k$.

Graafinväritysongelma on laskennallisesti hankala. Raja voidaan hakea ahneella algoritmilla: väritetään kukin solmu vuorollaan mahdollisimman pieninumeroisella värillä. Tai voidaan värittää graafin solmut aluksi, ja tarkastella kustakin C_l:stä, monenko värisiä solmuja se sisältää.

Haaraudu ja rajoita (branch and bound)

Aiemmin vaihtoehdot $x \in C_l$ on käyty läpi mielivaltaisessa järjestyksessä. Parempi voi olla laskea kullekin x:lle $B(X + [x])$ ja tarkastella ensin lupaavimpia vaihtoehtoja. Näin saatetaan löytää hyviä ratkaisuja karsimaan hakua.

```
BranchAndBound(X):
jos X on käypä ratkaisu:
    P ← profit(X)
        if P > OptP:
            OptP ← P
            OptX ← X

laske C_l

v ← [ ]
kullekin x ∈ C_l:
    laske B_x ← B(X + [x])
    v ← v + [(x, B_x)]
järjestä v B_x:n laskevaan järjestykseen

kullekin (x, B_x) ∈ v:
    if B_x ≤ OptP:
        return
    BranchAndBound(X + [x])
```

Dynaaminen ohjelmointi maksimiklikkiongelmassa

Etsitään graafin $G = (\mathcal{V}, \mathcal{E})$, missä $\mathcal{V} = \{1, \ldots, n\}$. Esilasketaan $N_v = \{u \in \mathcal{V} : \{u, v\} \in E\}$ ja $G_v = \{u \in \mathcal{V} : u \geq v\}$. Merkitään c_v:llä suurimman G_v:hen sisältyvän klikin S kokoa. Nyt $c_{i-1} \in \{c_i, c_i + 1\}$ ja $c_{i-1} = c_i + 1$ vain silloin, kun G_{i-1} sisältää $c_i + 1$ solmun klikin (johon väistämättä kuuluu solmu $i - 1$).

```
for i = n downto 1:
    found ← false
    MaxClique([i], G_i ∩ N_i)
    c_i ← OptP
MaxClique(X, N):
if |X| > OptP :
    OptP ← |X|
    OptX ← X
    found ←true
    return
if |X| + |N| ≤ OptP:
    return
for x ∈ N:
    if |X| + c_x ≤ OptP:
        return
    MaxClique(X + [x], N ∩ N_x)
    if found:
        return
```


Heuristiset menetelmät

heuristic: involving or serving as an aid to — problem-solving by experimental and especially trial-and-error methods; also : of or relating to exploratory problem-solving techniques that utilize self-educating techniques (as the evaluation of feedback) to improve performance

- kun ratkaisuavaruus on liian suuri peräytyvälle haulle
- etsivät hyviä ratkaisuja yrityksen ja erehdyksen kautta tekemällä muutoksia aiempiin ratkaisuihin
- sopivat optimointiongelmiin (jos hyvä ratkaisu riittää) ja hakuongelmiin, mutta eivät yleensä enumerointi- tai generointiongelmiin

Optimointiongelma

$$\begin{gathered} \max P(x) \\ g_j(x) \leq 0,\ j = 1 \ldots m \\ x \in X \\ X \text{ äärellinen} \end{gathered}$$

$P(x)$ on kohdefunktio, ja $g_j(x) \leq 0$:t ovat rajoitusehtoja. Mikä tahansa $x \in X$ on ratkaisu. Jos lisäksi $g_j(x) \leq 0$, x on käypä ratkaisu. Jos $P(x) \geq P(x')$ kaikilla $x' \in X$, $g_j(x') \leq 0$, ratkaisu x on optimaalinen.

Sakkofunktiomenetelmällä saadaan epäkäyvistä ratkaisuista käypiä, mikä voi helpottaa haun laatimista. Näin voidaan myös esim. esittää hakuongelma optimointiongelmana.

$$\begin{gathered} \max P(x) - \mu \textstyle\sum_j \Phi\left(g_j(x)\right) \\ x \in X \\ X \text{ äärellinen,} \end{gathered}$$

missä $\Phi(y) = 0$, kun $y \leq 0$, ja $\Phi(y) > 0$, kun $y > 0$. Voidaan valita riittävän suuri μ:n arvo heti aluksi tai kasvattaa μ:tä vähitellen.

Yleinen heuristinen menetelmä

Heuristisissa menetelmissä keskeinen käsite on naapuristo. Ratkaisun x naapuristo on $N(x) \subseteq X$. Heuristiikka $h_N(x)$ palauttaa jonkin käyvän ratkaisun x:n naapuristosta tai $Fail$.

```
GenericHeuristicSearch:
x ← jokin käypä x ∈ X
BestX ← x
while not lopetusehto:
    y ← h_N(x)
    if y ≠ Fail
        x ← y
        if P(x) > P(BestX)
            BestX ← X
return BestX
```

Mahdollisia yksinkertaisia heuristiikkoja

1. etsi käypä $y \in N(x) \setminus \{x\}$, jolle $P(y)$ on suurin; palauta y tai $Fail$, jos ainoatakaan käypää ratkaisua ei ole
2. etsi käypä $y \in N(x)$, jolle $P(y)$ on suurin; jos $P(y) > P(x)$, palauta y, muuten $Fail$
3. etsi jokin käypä $y \in N(x)$
4. etsi jokin käypä $y \in N(x)$; jos $P(y) > P(x)$, palauta y, muuten $Fail$

Tasainen graafin ositus

Annettu: $2n$ solmun täydellinen graafi $G = (\mathcal{V}, \mathcal{E})$ ja kustannusfunktio cost : $\mathcal{E} \mapsto \mathbb{Z}^+ \cup \{0\}$.

$$\min C\{\mathcal{V}_0, \mathcal{V}_1\} = \sum_{v_0 \in \mathcal{V}_0, v_1 \in \mathcal{V}_1} \text{cost}(\{v_0, v_1\})$$

$$\mathcal{V} = \mathcal{V}_0 \cup \mathcal{V}_1, |\mathcal{V}_0| = |\mathcal{V}_1| = n$$

Esimerkkiratkaisu: Ratkaisuavaruus X olkoon $\mathcal{V}$:n ositusten $[\mathcal{V}_0, \mathcal{V}_1]$, joille $|\mathcal{V}_0| = |\mathcal{V}_1| = n$. Osituksen x naapuristoksi $N(x)$ valitaan niiden ositusten joukko, jotka saadaan x:stä siirtämällä kummastakin joukosta yksi alkio toiseen. Heuristiikka $h_N(x)$ voisi olla steepest ascent: etsi paras naapuri y; jos kohdefunktio paranee, palauta y, muuten $Fail$.

Naapuristoheuristiikat

Maksimoitaessa kullakin iteraatiolla

Steepest ascent: Valitaan x:n käypä naapuri, jolla kohdefunktion arvo on suurin, kunnes ei enää löydy naapuria, johon siirryttäessä kohdefunktio kasvaisi

Hill-climbing: Valitaan jokin x:n käypä naapuri, jolla kohdefunktion arvo kasvaa, kunnes sellaista ei enää löydy

Molemmat näistä pysähtyvät ensimmäiseen paikalliseen optimiin. Paikallinen optimi on ratkaisu x, jolle pätee, että $P(x) > P(y)$ kaikilla käyvillä $y \in N(x)$.

Lokaaleja optimeja voidaan jossakin määrin vähentää laajentamalla naapuristoa, mutta se harvoin poistaa ongelman; lisäksi naapuriston kasvaessa $h_N(x)$:n laskenta tulee työläämmäksi.

Naapuristoheuristiikat II

Great deluge: Valitaan joka iteraatiolla käypä naapuri $y \in N(x)$, jolle $P(y) \geq W$, missä W on vedenpinnan taso. Kasvatetaan W:tä aika ajoin, kunnes käypiä naapureita ei ole.

Record-to-record travel: Valitaan joka iteraatiolla käypä naapuri $y \in N(x)$, jolle $P(y) \geq P(BestX) - D$, missä D on vakio.

Simuloitu jäähdytys

Simuloitu jäähdytys perustuu analogiaan metallin jäähtymisestä.

$h_N(x)$: Valitaan satunnainen $y \in N(x)$. Olkoon $\Delta P = P(y) - P(x)$. Jos $\Delta P \geq 0$, palautetaan y. Jos $\Delta P < 0$, palautetaan y todennäköisyydellä $e^{\Delta P/T}$, missä T on systeemin senhetkinen lämpötila; muuten $Fail$.

Aluksi T on suhteellisen suuri, joten ratkaisua huonontavia siirtoja hyväksytään usein. Haun mittaan lämpötilaa lasketaan, jolloin huonontavia siirtoja hyväksytään yhä harvemmin.

Yksinkertaisimmillaan voidaan joka iteraatiolla asettaa $T \leftarrow \alpha T$, missä $\alpha < 1$, mutta $\alpha \approx 1$.

Tabu-haku

Heuristiikka, jossa valitaan x:n käypä naapuri, jolla kohdefunktion arvo on suurin, siirtyy kyllä pois paikallisesta optimista, mutta palaa usein heti takaisin. Siksi tabu-haku:

$h_N(x)$: Valitaan x:n käypä naapuri, jolla kohdefunktion arvo on suurin, kuitenkin perumatta mitään viimeisen l iteraation aikana tehtyä muutosta.

change (x, y) kuvaa muutosta, joka tehdään siirryttäessä x:stä sen naapuriin y:hyn.

TabuSearch

$TabuList \leftarrow [\,]$
valitse käypä $x \in X$
while not $lopetusehto$:
    $T = \{y : y \in N(x)$, change $(x, y) \in TabuList\}$
    $N \leftarrow N(x) \setminus T$
    etsi käypä $y \in N$ jolle $P(y)$ on suurin
    $x \leftarrow y$
    lisää $TabuList$:iin change (y, x)
    poista $TabuList$:stä kaikki paitsi l uusinta alkiota
    if $P(X) > BestP$
        $BestP \leftarrow P(X)$
        $BestX \leftarrow X$

Naapuriston valinta

Naapuriston ja ratkaisuavaruuden yhdistelmä voidaan tulkita (suunnattuna) verkkona, jonka solmut ovat ratkaisuja. Solmusta x on kaari solmuun y, jos $y \in N(x)$.

Hyvän naapuriston ominaisuuksia:

1. Jokainen ratkaisu—tai ainakin optimiratkaisu—on saavutettavissa mistä tahansa ratkaisusta.
2. Naapuristo on suhteellisen pieni, ja ratkaisun ja sen naapurien kohdefunktioiden arvot korreloivat edes jossakin määrin.
3. Naapuristo on riittävän suuri, jotta jokainen ratkaisu—tai ainakin optimiratkaisu—on saavutettavissa mistä tahansa ratkaisusta pienehköllä määrällä siirtoja.

Kohdefunktion sileys

On huomattavaksi eduksi, jos kohdefunktion arvo on suuri ratkaisuilla, jotka ovat lähellä optimiratkaisua. Olkoon $x_i \in \{0,1\}$ ja
$N(x) = \{y \in \{0,1\}^n : \text{dist}(x,y) = 1\}$.

Vrt.
1. $\max \sum_i x_i$
2. $\max \prod_i x_i$
3. $\max \sum_i x_i - 3\,(\sum_i x_i \mod 4)$
4. $\max 2n \prod_i x_i - \sum_i x_i$

Laajat "laaksot" tai "tasangot" kohdefunktiossa ovat usein hankalia. Valittu ongelman esitystapa ja naapuristo ovat avainasemassa.

Geneettiset algoritmit

Sen sijaan, että ylläpidettäisiin yhtä senhetkistä ratkaisua, voidaan ylläpitää kokonaista populaatiota. Uusia ratkaisuja saadaan risteyttämällä vanhoja ja mutatoimalla niitä.

Risteytyksessä kahdesta ratkaisusta saadaan kaksi uutta ratkaisua. Mutaatiossa ratkaisu x korvataan jollakin naapurillaan; $x \leftarrow h_N(x)$.

GeneticAlgorithm

valitse alkupopulaatio P
while not $lopetusehto$:
    $Q \leftarrow P$
    laske risteytettävien parien lista R
    for $(w,x) \in R$
        (y,z) = risteytys (w,x)
        $y \leftarrow h_N(y)$
        $z \leftarrow h_N(z)$
        $Q \leftarrow Q \cup \{y,z\}$
        $P \leftarrow Q$:n $popsize$ parasta yksilöä
    $b \leftarrow Q$:n paras yksilö
    if $P(b) > P(BestX)$
        $BestX \leftarrow b$

Risteytys ja parinmuodostus

Listat $A = [a_1, \ldots, a_n]$ ja $B = [b_1, \ldots, b_n]$ voidaan risteyttää esim. seuraavasti:

single-point crossover Valitaan $1 \le j < n$. Jälkeläiset ovat
$$C = \left[a_1, \ldots a_j, b_{j+1}, \ldots b_n\right] \text{ ja}$$
$$D = \left[b_1, \ldots b_j, a_{j+1}, \ldots a_n\right].$$

two-point crossover Valitaan $1 \le j < k \le n$. Jälkeläiset ovat
$$C = \left[a_1, \ldots, a_j, b_{j+1}, \ldots, b_k, a_{k+1}, \ldots, a_n\right] \text{ ja}$$
$$D = \left[b_1, \ldots, b_j, a_{j+1}, \ldots, a_k, b_{k+1}, \ldots, b_n\right].$$

uniform crossover Valitaan $S \subseteq \{1, \ldots, n\}$. Jälkeläisillä $c_i = a_i$ ja $d_i = b_i$, jos $i \in S$; muuten $c_i = b_i$ ja $d_i = a_i$.

Permutaatioiden α ja β risteyttämiseksi voidaan valita $1 \le j < k \le n$, ja

PartiallyMatchedCrossover(n, α, β, j, k)
$\gamma \leftarrow a$
$\delta \leftarrow \beta$
for $i = j$ to k:
    $\gamma \leftarrow (a_i\ \beta_i)\,\gamma$
    $\delta \leftarrow (a_i\ \beta_i)\,\delta$

Risteytyksen tulokset eivät aina ole käypiä ratkaisuja. Voidaan
1) soveltaa sakkofunktiomenetelmää
2) räätälöidä ongelmalle risteytysfunktio, jolla jälkeläiset ovat aina käypiä.

Parinmuodostus: populaation ratkaisut voidaan parittaa eri kriteerein, esim. paremmuusjärjestyksessä. Voidaan myös generoida hyvistä ratkaisuista enemmän jälkeläisiä.

Steinerin kolmikkojärjestelmät

Steinerin kolmikkojärjestelmä on joukkojärjestelmä $(\mathcal{V}, \mathcal{B})$, missä $\mathcal{B}$ koostuu $\mathcal{V}$:n 3-osajoukoista, ja kukin $\mathcal{V}$:n 2-osajoukko on täsmälleen yhden $b \in B$:n osajoukko. STS on täydellisen graafin kaarien ositus kolmioiksi. Esim.

$$\begin{aligned} \mathcal{V} &= \{1, \ldots 7\} \\ \mathcal{B} &= \left\{ \begin{array}{lll} \{1,2,4\}, & \{2,3,5\}, & \{3,4,6\}, \\ \{4,5,7\}, & \{1,5,6\}, & \{2,6,7\}, \\ \{1,3,7\} & & \end{array} \right\} \end{aligned}$$

Hill-climbing ja Steinerin kolmikkojärjestelmät

Piste $v \in V$ on *elävä*, jos siitä lähtee kaaria, jotka eivät vielä ole missään kolmiossa $b \in B$. Kaari $\{u, v\}$ on elävä, jos se ei esiinny missään kolmiossa $b \in B$.

Stinson'sAlgorithm(v):
$\mathcal{V} \leftarrow \{1, \ldots, n\}$
$\mathcal{B} \leftarrow \emptyset$
while $|\mathcal{B}| < v\,(v-1)\,/6$:
    valitse elävä piste x
    valitse elävät kaaret $\{x, y\}$ ja $\{x, z\}$
    if $\{y, z\}$ on elävä:
        $\mathcal{B} \leftarrow B \cup \{\{x, y, z\}\}$
    else
        etsi $\mathcal{B}$:stä blokki $\{w, y, z\}$
        $\mathcal{B} \leftarrow B \cup \{\{x, y, z\}\} \setminus \{\{w, y, z\}\}$

Selkäreppu ja simuloitu jäähdytys

Selkäreppuongelma: Annetaan n esinettä, joiden painot ovat $w_1, \ldots w_n$ ja hyödyt $p_1, \ldots, p_n$. Repun kapasiteetti on M. Maksimoidaan $P(x) = \sum p_i x_i$ rajoitusehdoilla $x_i \in \{0, 1\}$ ja $\sum w_i x_i \le M$.

Valitaan naapuristo:
$N(x) = \{y \in \{0,1\}^n : \text{dist}\,(x, y) = 1\}$.
Satunnainen naapuri y saadaan kääntämällä satunnainen x_j.

Jos $x_j = 0$, kohdefunktion muutos $\Delta P = +p_j$, ja uusi naapuri hyväksytään, jos se on käypä. Jos $x_j = 1$, $\Delta P = -p_j$, ja uusi naapuri hyväksytään todennäköisyydellä $e^{p_j/T}$.

Valitaan aluksi T siten, että suuri osa huonontavistakin siirroista hyväksytään; esim. $4 \max_i p_i$, ja asetetaan joka iteraation jälkeen $T \leftarrow \alpha T$. Kirjassa parhaat tulokset saatiin $\alpha = 0.9999$:llä.

Selkäreppu ja tabu-haku

Valitaan naapuristo:
$N(x) = \{y \in \{0,1\}^n : \text{dist}(x,y) = 1\}$.

Ei maksimoidakaan hyötyfunktiota(?!!), vaan

1. lisätään reppuun esine i, joka ei ole tabu-listalla ja jolla on suurin p_i/w_i-suhde niistä esineistä, joilla $x_i = 0$ ja jotka mahtuvat reppuun

2. ellei sellaista ole, poistetaan repusta esine i, joka ei ole tabu-listalla ja jolla on pienin p_i/w_i-suhde niistä esineistä, joilla $x_i = 1$.

3. Lisätään tabulistaan i.

Graafin väritys

Mikä on pienin määrä värejä, joilla graafi $G = (\mathcal{V}, \mathcal{E})$ voidaan värittää siten, että u ja v ovat erivärisiä, jos $\{u,v\} \in \mathcal{E}$?

Ositetaan solmut väriluokkiin $\mathcal{V}_1 \ldots \mathcal{V}_k$ esim. ahneella algoritmilla. Kun on löydetty k-väritys, etsitään heuristisella haulla $k-1$-väritystä seuraavasti.

Otetaan kohdefunktioksi $\max \sum_i |\mathcal{V}_i|^2$. Tämä ohjaa hakua sellaiseen suuntaan, että joillakin väreillä väritetään paljon solmuja (ja toisilla taas vähän). Jos jonkun osan koko menee nollaan, on löytynyt $k-1$-väritys, ja voidaan alkaa etsiä $k-2$-väritystä jne.

Schurin luvut

Schurin luku $s(m)$ on suurin kokonaisluku s, jolla kokonaisluvut $X = \{1,\ldots,s\}$ voidaan osittaa m osaan siten, että missään joukossa ei ole kahta (ei välttämättä eri) alkiota ja niiden summaa.

Esim. $s(3) = 13$: $\left\{ \begin{array}{l} \{1,4,7,10,13\} \\ \{2,3,11,12\} \\ \{5,6,8,9\} \end{array} \right\}$

Kun etsitään lukujen $1 \ldots s$ summatonta ositusta, ilmeisin valinta kohdefunktioksi olisi eri osissa esiintyvien summien lukumäärä, mutta parempi on maksimoida $\max c_1 f_1 + c_2 f_2$, missä $c_1 \gg c_2$ ja $f_1 = \max t$, jolla missään osassa ei ole summaa t
$f_2 = \sum_{i=1}^{m} \sum_{t,u \in X_i} g(t,u)$

missä $g(t,u) = \left\{ \begin{array}{ll} 0 & \text{jos } t+u \leq s \\ 2s-t-u & \text{jos } t+u > s \end{array} \right.$

Tässä f_2:n tarkoitus on vain ohjata hakua lupaavaan suuntaan.

Kauppamatkustajan ongelma

Ratkotaan kauppamatkustajan ongelmaa geneettisillä algoritmeilla.

n-opt-naapuristo: poistetaan syklistä n kaarta ja lisätään siihen n kaarta niin, että saadaan taas sykli.

Voidaan määritellä risteytysfunktio seuraavasti: Risteytetään kaksi ratkaisua jollakin permutaatioita yhdistelevällä risteytysfunktiolla, ja sovelletaan tämän jälkeen saatuun tulokseen steepest descent -menetelmää esim. 2-opt-naapuristolla.

Voidaan myös yksinkertaisesti risteyttää permutaatiot ja lisätä steepest descent kohdefunktioon. Tällöin permutaation hyvyyttä arvioitaisiin soveltamalla siihen ensin steepest descent -hakua ja laskemalla kohdefunktion arvo tämän jälkeen.

Isomorphisms

Labeled structures are usually not considered significantly different, if the only difference is the labeling.

An isomorphism is a bijection that maps the parts of one structure onto the parts of another structure so that the structure is preserved. Two structures are isomorphic, if there is an isomorphism from one to the other.

Example

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, if there exists a bijection $f : V_1 \mapsto V_2$ s.t. $\{u, v\} \in E_1$ if and only if $\{f(u), f(v)\} \in E_2$.

Automorphisms

An automorphism is an isomorphism from a structure onto itself. In a sense, automorphisms represent the symmetries of the structure.

Example

An automorphism of the graph $G = (V, E)$ is a bijection (permutation) $\pi : V \mapsto V$, for which $\{u, v\} \in E$ if and only if $\{\pi(u), \pi(v)\} \in E$.

The permutations of the vertex set form a *group*, and the automorphisms of a graph form a *subgroup* of this group.

Group

A set-operation pair $(G, *)$ is a group, if

1. the binary operation $*$ is closed: $g_1 * g_2 \in G$ for all $g_1, g_2 \in G$, i.e., $* : G \times G \mapsto G$
2. G contains a unit element $\mathbf{I}$, s.t. $g * \mathbf{I} = g = \mathbf{I} * g$ for all $g \in G$
3. every $g \in G$ has the inverse element $g^{-1} \in G$, such that $g^{-1} * g = \mathbf{I} = g * g^{-1}$
4. the binary operation $*$ on associative: $(g_1 * g_2) * g_3 = g_1 * (g_2 * g_3)$ for all $g_1, g_2, g_3 \in G$

Examples

- integers modulo n under addition
- $m \times m$-matrices with nonzero determinant under matrix multiplication
- rotations of a three-dimensional object

When the operation is obvious from the context, we speak of the group G.

Multiplication table

A finite group may be presented as a multiplication table:

$*$	$\mathbf{I}$	a	b	c	d	e	f	g
$\mathbf{I}$	$\mathbf{I}$	a	b	c	d	e	f	g
a	a	b	c	$\mathbf{I}$	e	f	g	d
b	b	c	$\mathbf{I}$	a	f	g	d	e
c	c	$\mathbf{I}$	a	b	g	d	e	f
d	d	g	f	e	$\mathbf{I}$	c	b	a
e	e	d	g	f	a	$\mathbf{I}$	c	b
f	f	e	d	g	b	a	$\mathbf{I}$	c
g	g	f	e	d	c	b	a	$\mathbf{I}$

When we place the unit element $\mathbf{I}$ in the first row and column, the multiplication table is a reduced Latin square with associativity $M\left[M\left[g_i, g_j\right], g_k\right] = M\left[g_i, M\left[g_j, g_k\right]\right]$.

Subgroup:
H is a subgroup of G, if H is a group and $H \subseteq G$.
For example, $\{\mathbf{I}, a, b, c\}$ in the group given above.

Of subgroups

The order $|G|$ of a finite group G is the number of its elements.

If H is a nonempty subset of a finite group G and H is closed under the operation of G, then H is a subgroup of G.

Proof. If $H = \{\mathbf{I}\}$, then H is clearly a subgroup. Suppose that $h_1 h_2 \in H$ for all $h_1, h_2 \in H$. Let us choose some $h \in H$. For all $n \in \mathbb{Z}^+$ it holds that $h^n = hh \ldots h \in H$. Since H is finite, there must exist some $m < n$ s.t. $h^m = h^n$. Now $h^n h^{n-m} = h^m h^{n-m} = h^n$, so $h^{n-m} = \mathbf{I} \in H$ and $h^{-1} = h^{n-m-1} \in H$.

Permutation groups

Let us consider the permutations of a finite set $\mathcal{X}$.

The permutations (bijektiot $\pi : \mathcal{X} \mapsto \mathcal{X}$) form a group under function composition
$(\pi_1 \pi_2)(x) = (\pi_1 \circ \pi_2)(x) = \pi_1(\pi_2(x))$, since

1. the composition of two permutations is a permutation
2. there is an identity element ($\mathbf{I}(x) = x$)
3. every permutation has an inverse permutation
4. function composition is associative

Let $\mathcal{X}$ be a nonempty set with n elements and Sym$(\mathcal{X})$ the set of its permutations. Sym$(\mathcal{X})$ under function composition is the symmetric group Sym(X). over the elements of X. It has $n!$ elements.

Every permutation group is a subgroup of some symmetric group. For example when $\mathcal{X} = \{0,1,2,3,4\}$, the permutations $\{\mathbf{I}, (0,1,2)(3,4), (0,2,1), (3,4), (0,1,2), (0,2,1)(3,4)\}$ form a permutation group over X.

Automorphisms of a graph

Let us denote $\alpha(\{u,v\}) = \{\alpha(u), \alpha(v)\}$

An automorphism α of a graph $G = (\mathcal{V}, \mathcal{E})$ is such a permutation of $\mathcal{V}$ that $\alpha(\{u,v\}) \in \mathcal{E}$ for all edges $\{u,v\} \in \mathcal{E}$ of the graph.

The automorphisms form the group Aut(G):

Aut(G) is nonempty, since clearly $\mathbf{I} \in$ Aut(G)

If $\alpha, \beta \in$ Aut(G), then $\alpha\beta \in$ Aut(G): suppose that $\{u,v\} \in \mathcal{E}$. $\beta(\{u,v\}) \in \mathcal{E}$, and $(\alpha\beta)(\{u,v\}) = \alpha(\beta(\{u,v\})) \in \mathcal{E}$, so Aut$(G)$ is closed under composition.

Aut(G) is a nonempty subset of Sym$(\mathcal{V})$ that is closed under the same operation, so Aut(G) is a subgroup of Sym$(\mathcal{V})$ and therefore a group.

Generators

The elements $\alpha_1, \ldots, \alpha_r$ generate the group G, if every element $g \in G$ can be expressed as a finite product

$$g = \alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_m},$$

where $1 \le i_j \le r$ for all j. The elements $\alpha_1, \ldots, \alpha_r$ are generators for G, denoted with $G = \langle \alpha_1, \ldots, \alpha_r \rangle$.

Example: Rubik's cube

Ideal Toy Company stated on the package of the original Rubik cube that there were more than three billion possible states the cube could attain. It's analogous to MacDonald's proudly announcing that they've sold more than 120 hamburgers. (J. A. Paulos, Innumeracy)

```
              +--------------+
              |  1    2    3 |
              |  4  top    5 |
              |  6    7    8 |
+-------------+--------------+--------------+--------------+
|  9   10   11 | 17   18   19 | 25   26   27 | 33   34   35 |
| 12  left  13 | 20 front  21 | 28 right  29 | 36  rear  37 |
| 14   15   16 | 22   23   24 | 30   31   32 | 38   39   40 |
+-------------+--------------+--------------+--------------+
              | 41   42   43 |
              | 44 bottom 45 |
              | 46   47   48 |
              +--------------+
gap> cube := Group(
( 1, 3, 8, 6)( 2, 5, 7, 4)( 9,33,25,17)(10,34,26,18)(11,35,27,19),
( 9,11,16,14)(10,13,15,12)( 1,17,41,40)( 4,20,44,37)( 6,22,46,35),
(17,19,24,22)(18,21,23,20)( 6,25,43,16)( 7,28,42,13)( 8,30,41,11),
(25,27,32,30)(26,29,31,28)( 3,38,43,19)( 5,36,45,21)( 8,33,48,24),
(33,35,40,38)(34,37,39,36)( 3, 9,46,32)( 2,12,47,29)( 1,14,48,27),
(41,43,48,46)(42,45,47,44)(14,22,30,38)(15,23,31,39)(16,24,32,40));;
gap> Size( cube );
43252003274489856000
```


Cosets

When $g \in G$ and $H \subseteq G$, we denote $gH = \{gh : h \in H\}$. When $A, B \subseteq G$, we denote $AB = \{ab : a \in A, b \in B\}$.

Let H be a subgroup of a finite group G. Now gH is the left coset of G that contains $g \in G$.

Lagrange: if H is a subgroup of G, then the elements of G may be partitioned into disjoint cosets:

$$G = g_1H \cup g_2H \cup \ldots \cup g_nH,$$

where $g_i \in G$, and $g_iH \cap g_jH = \emptyset$ for $i \neq j$.

Proof. $|gH| = |H|$ for all $g \in G$, since $f(x) = gx$ is a bijection $H \mapsto gH$. If $g_1H \cap g_2H \neq \emptyset$, where $g_1, g_2 \in G$, there must exist some $h_1, h_2 \in H$, for which $g_1h_1 = g_2h_2$ and $g_1 = g_2h_2h_1^{-1}$. Now for any $h \in H$ we have $g_1h = g_2\left(h_2h_1^{-1}h\right) \in g_2H$, and $g_1H \subseteq g_2H$. Since $|g_1H| = |g_2H| = |H|$, $g_1H = g_2H$. Additionally each $g \in G$ belongs to the coset gH; the cosets therefore partition G and $|G|$ is divisible with $|H|$.

Transversals

When G is presented as a union of disjoint left cosets

$$G = g_1H \cup g_2H \cup \ldots \cup g_nH,$$

the set $T = \{g_1, \ldots, g_n\}$ forms a left transversal of H. It can be formed by selecting $n = \frac{|G|}{|H|}$ coset representatives $g_i \in G$ such that no chosen g_i belongs to any coset other than g_iH.

Transversals: example

$*$	$\mathbf{I}$	a	b	c	d	e	f	g
$\mathbf{I}$	$\mathbf{I}$	a	b	c	d	e	f	g
a	a	b	c	$\mathbf{I}$	e	f	g	d
b	b	c	$\mathbf{I}$	a	f	g	d	e
c	c	$\mathbf{I}$	a	b	g	d	e	f
d	d	g	f	e	$\mathbf{I}$	c	b	a
e	e	d	g	f	a	$\mathbf{I}$	c	b
f	f	e	d	g	b	a	$\mathbf{I}$	c
g	g	f	e	d	c	b	a	$\mathbf{I}$

The subgroup $H = \{\mathbf{I}, a, b, c\}$ has the cosets $\{\mathbf{I}, a, b, c\}$ and $\{d, e, f, g\}$. A transversal can be formed by choosing an element from each coset; e.g., $T = \{\mathbf{I}, d\}$ or $T = \{b, f\}$.
$TH = \mathbf{I}\{\mathbf{I}, a, b, c\} \cup d\{\mathbf{I}, a, b, c\} = \{\mathbf{I}, a, b, c\} \cup \{d, e, f, g\} = G$.

Computing a transversal

By computing a transversal T of $G_{\mathcal{B}} \subseteq G$ and then $T(\mathcal{B})$ we obtain the orbit $G(\mathcal{B})$ of $\mathcal{B}$.

Below a naive method for computing a transversal is given. For each element of the group we test whether our transversal already contains an element from the same coset. If not, we add the element to the transversal.

Transversal(H, G):
$r \leftarrow |G| \,/\, |H|$
$T \leftarrow \emptyset$
for $g \in G$:
    for $t \in T$:
        if $t^{-1}g \in H$:
            goto skip
    $T \leftarrow T \cup \{g\}$
    if $|T| \geq m$:
        return T
    skip:

Group action

The action of a group G on the set $\mathcal{X}$ is a function $\alpha : G \times \mathcal{X} \mapsto \mathcal{X}$, denoted with $\alpha : (g, x) \mapsto gx$, that satisfies

1. $\mathbf{I}x = x$ for all $x \in \mathcal{X}$
2. $g(hx) = (gh)x$ for all $g, h \in G$ and $x \in \mathcal{X}$.

Note that if $gx_1 = gx_2$, then
$g^{-1}(gx_1) = (g^{-1}g)x_1 = \mathbf{I}x_1 = x_1$
$= g^{-1}(gx_2) = (g^{-1}g)x_2 = \mathbf{I}x_2 = x_2$.

In fact each g defines a permutation of $\mathcal{X}$.

Group action. Example: graph

When discussing the symmetric group, we usually speak of the group S_n, whose structure is the same as that of the group formed by the permutations of the set $\{1, \ldots, n\}$. When S_n acts on a set V with n elements just like the permutations of V, we say that S_n acts on V in the natural way.

Now, say that a group acts on the vertices of a graph $G = (V, E)$ in some way, then the group acts in the *induced manner* on the edges of the graph. In fact, this also induces an action on the set of graphs.

Group action. Example: binary code

Two binary codes (sets of binary codewords of the same length) can be considered equivalent, if one can be obtained from the other by complementing all bits in certain positions in the codewords and permutating the positions.

This corresponds to the action of a group that is the wreath product $S_2 \wr S_n$, where the action of S_n corresponds to permuting the positions in the codewords and the action of each S_2 (of which there are n) corresponds to complementing the bits in a given position.

(We will not examine the characteristics of the wreath product.)

Group action. Example: dihedral group

The elements of the dihedral group D_n correspond to the symmetries of a regular n-gon. It may be defined as follows: $D_n = \langle r, s \rangle$, where $r^n = s^2 = (rs)^2 = \mathbf{I}$; the group has two generators, and the given constraints uniquely determine the structure of the group (when we assume that the given exponents are the least ones with which the identity element is obtained). Here r corresponds to a $1/n$ rotations clockwise and s to mirroring across some axis.

D_n can act on a set $V = \{0, \ldots, n-1\}$ for example as follows: $rv = (v+1) \bmod n$, $sv = (n-v) \bmod n$.

D_n can act on $\mathbf{R}^2$ as follows: $rx = \begin{pmatrix} \cos 2\pi/n & -\sin 2\pi/n \\ \sin 2\pi/n & \cos 2\pi/n \end{pmatrix} x$,

$sx = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} x$

The orbit of an element

The orbit of an element x is $G(x) = \{gx : g \in G\} \subset \mathcal{X}$ and the stabilizer of x is $G_x = \{g \in G : gx = x\} \subseteq G$. Since G_x is nonempty ($\mathbf{I} \in G_x$) and closed, it is a subgroup, and we can find a transversal. If for two elements g_i and g_j in the transversal it holds that $g_i x = g_j x$, then $g_i^{-1} g_i x = g_i^{-1} g_j x = x$, and $g_i^{-1} g_j \in G_x$. Now $g_i^{-1} g_j G_x = G_x$ and $g_i G_x = g_i g_i^{-1} g_j G_x = g_j G_x$ – but the transversal only contains one element from each coset, so $g_i = g_j$. Therefore $|G(x)| = |G| \,/\, |G_x|$.

Searching for orbit representatives

When we know N_{k+1}, the number of orbits of $k+1$-element subsets, we can find one set from each orbit as follows. If $\mathcal{R}$ is a set of orbit representatives of k-subsets,

$$S = \{A \cup \{x\} : A \in \mathcal{R}, x \in X \setminus A\}$$

will contain at least one (maybe more) representative from each $k+1$-subset orbit. Representatives from the same orbit must be removed until we only have one representative from each orbit.

A simple[1] idea:

for all $g \in G$:
  for all $A \in S$ in decreasing lex. order:
    if $\mathrm{rank}(g(A)) < \mathrm{rank}(A)$:
      $S \leftarrow S \cup \{g(A)\} \setminus \{A\}$
      if $|S| = N_{k+1}$:
        return

[1] and wrong: consider $G = \{\mathbf{I}, (1,2)(3,4), (1,4)(2,3), (1,3)(2,4)\}$ and $S = \{\{1\}, \ldots, \{4\}\}$. Perhaps union-find or something?

Orderly algorithm

When a group G acts on a totally ordered set X, and on the subsets of X in the induced way, we can order the k-subsets as follows: $S \prec T$, if there is an $s \in S$, for which $s \notin T$, and for all $x \prec s$ either $x \in S$ and $x \in T$ or $x \notin S$ and $x \notin T$.

Starting from the empty set, we can obtain the minimum representatives of the subset orbits by the following algorithm:

orderly(S):
  process S
  $C = \{x : x \in X \wedge x > s \; \forall s \in S\}$
  for x in C:
    if canonical($S \cup \{x\}$):
      orderly($S \cup \{x\}$)

Proof: We denote $F(S) = S \setminus \{\max S\}$. F is weakly monotonic: $S_1 < S_2 \Rightarrow F(S_1) \leq F(S_2)$.

Base case of induction: When $n = 0$, all canonical n-elements subsets are processed.

Induction step: If all canonical n-subsets are processed, then also all canonical $n+1$-subsets are processed. Let S be a canonical $n+1$-subset. Since S is canonical, $S \leq g(S)$ for all $g \in G$, and $F(S) \leq F(g(S))$ for all $g \in G$. We find that $F(g(S)) \leq g(F(S))$ for all $g \in G$ — both are obtained by removing one element from $g(S)$, in case of $F(g(S))$ the element $\max g(S)$. Since $F(S) \leq g(F(S))$ for all $g \in G$, $F(S)$ is canonical. Thus by induction S is processed, since $F(S)$ is canonical.

Orderly algorithm. Example I

Sum packing $\bmod n$: For a given n we find a maximum set $S \subseteq \mathbb{Z}_n$, for which no $x \in \mathbb{Z}_n$ can be presented as two different sums of two elements of S. It is easy to define an order on the elements of $\mathbb{Z}_n$, and this defines the lexicographical order of the k-subsets.

Functions of the form $f(x) = ax + b \pmod n$ preserve equal sums as equal and distinct sums as distinct, as long as $\gcd(a, n) = 1$. These functions form a group. The canonicity test for a subset S can be performed by testing for all elements f in the group, whether $f(S) < S$.

Orderly algorithm. Example II

A binary code is a set of n-bit binary words. In a minimum distance code each pair of codewords must differ in at least d positions for some d. Equivalence: the bit positions can be permutated freely, and the bits in some position may be flipped. These distance-preserving operations define a group that acts on the set of codewords; when an order has been defined on the set of codewords, a lexicographical order can be defined on the codes.

It can be shown that the canonicity test can be performed as follows: consider the code as a 0/1-matrix, whose rows are codewords and columns represent bit positions. We use backtracking search to examine all possible permutations of the rows. For each permutation, we flip the bits in those positions where the first codeword has a 1, and then we sort the columns into ascending order. The lexicographically first code obtained is the canonical representative.

The canonical parent method

Isomorph representatives of structures can be constructed as follows: partition the structures to levels. When isomorph representatives of structures at level n (the parents) have been constructed, isomorph representatives of the structures at level $n+1$ (the children) can be constructed as follows:

From each parent, construct some set of children. The problem is that some children can end up being created several times 1) from different parents 2) from the same parent.

1. For each child, we define the canonical parent, i.e., the structure of level n from which it must be constructed, and during the search we check that the child has been constructed from the canonical parent. We must make sure that each child can be created from its canonical parent.
2. Carry out isomorph elimination for children from the same parent.

For each level n structure p in turn we construct a set Q of level $n+1$ structures. For each $q \in Q$ we compute $F(q) = p'$, a level n structure. F must preserve isomorphism: if $q_1 \cong q_2$, then $F(q_1) \cong F(q_2)$. We reject those $q \in Q$ for which p ja p' are not isomorphic ($p \ncong p'$). From the remaining elements in Q we eliminate duplicates so that exactly one element from each isomorph class remains.

Canonical parent method for graphs

Nonisomorphic graphs can be constructed with the canonical parent method as follows. Level n structures are the graphs with n vertices. Let f be a function that removes the vertex with the highest number from a labeled graph. Let c be a function that computes the canonical form of a graph. We can define the canonical parent function as $F(G) = f(c(G))$.

The only level 1 graph has 1 vertex and no edges. From level n graphs we can construct the level $n+1$ graphs as follows: Examine each level n graph G in turn. From G form the graphs which can be obtained by adding a vertex v and zero or more edges with v as an endpoint. For each graph H thus obtained compute the canonical parent: $G' = F(H)$. If $G \ncong G'$ — we may e.g. test if $c(G) \neq c(G')$ — reject H. Carry out isomorph rejection for the children and move on to the next level n graph.

If every isomorph class of n-vertex graphs is represented, then each $n+1$-vertex isomorph class will be represented, since for every $n+1$-vertex graph H there is a graph H' isomorphic to H that can be generated from a graph isomorphic to $f(c(H))$.

The canonical augmentation method

The canonical augmentation method is a stronger version of the canonical parent method. In the canonical parent method, when structure q is constructed from parent p, we test whether $F(q) \cong p$. In the canonical augmentation method, we instead consider whether the augmentation (q,p) is isomorphic to the canonical augmentation $(q,F(q))$.

Now we require of F that $q_1 \cong q_2 \implies (q_1,F(q_1)) \cong (q_2,F(q_2))$. That is, if q_1 and q_2 are isomorphic, then some group element must map q_1 to q_2 and $F(q_1)$ to $F(q_2)$.

Suppose that $q_1 \cong q_2$, and both pass the augmentation test. Since F must map isomorphic children to isomorphic parents, $F(q_1)$ and $F(q_2)$ are isomorphic, and if isomorph testing has been properly carried out on the previous levels, q_1 and q_2 been generated from the same parent p. Then

$$(q_1,p) \cong (q_1,F(q_1)) \cong (q_2,F(q_2)) \cong (q_2,p),$$

so some automorphism of p must map q_1 to q_2. It thus suffices to consider automorphisms of the parent to prune isomorphs from among its children.

Canonical augmentation method for graphs

The canonical augmentation method for graphs proceeds almost like the canonical parent method.

Let F be a function that chooses a vertex from a graph in a permutation-invariant manner.

When we have constructed a graph G from its parent $G \setminus \{v\}$ by adding a vertex v and edges with v an endpoint, we test whether $(G, G \setminus \{v\}) \cong (G, G \setminus F(G))$.

In practise we may take two copies of G, color v in one and $F(G)$ in the other with a distinct color, and test if they are isomorphic (the isomorphism must preserve the coloring).

To filter duplicates from the children of a parent, we need not necessarily store them in a list. We accept the child only if it is the lexicographical minimum representative of its orbit; it suffices to test the automorphism group of the parent. In particular, if the automorphism group of the parent is trivial, no pruning is necessary.

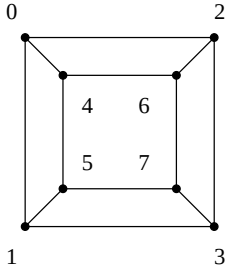
Computer representations of a permutation group

Computer representations of a permutation group should have the following properties:

1. We can check whether some permutation g is in the group G
2. We can list the elements of the group
3. The space requirements are reasonable

Example

Automorphisms of the cube graph



Aut $(G) = \langle \alpha, \beta \rangle$, where
$\alpha = (0,1,3,7,6,4)\,(2,5)$ and
$\beta = (0,1,3,2)\,(4,5,7,6)$.
$|G| = 48$.

Computer representations of permutation groups

We could store the permutations in the group, for example in lexicographical order.

1. We can determine by binary search whether $g \in G$.
2. We can easily list the elements
3. We need a lot of space; Sym (n) has $n!$ permutations

Computer representations of a permutation group

We could store only some set of generators for the group.

3. We need little space, but

1.–2. We must carry out a (say) breadth-first search to generate all elements, and we will get duplicates; in our example $\alpha\alpha\alpha\alpha\alpha\beta\beta\beta = \alpha\beta\alpha\alpha$.

Simplegen(Γ):
$G \leftarrow \emptyset$
$N \leftarrow \{\mathbf{I}\}$
while $N \neq \emptyset$:
  $G \leftarrow G \cup N$
  $N \leftarrow N\Gamma \setminus G$
where Γ is the set of generators and $N\Gamma$ is $\{ng : n \in N, g \in \Gamma\}$.

Schreier-Sims

Let G be a permutation group over $\mathcal{X} = \{0, \ldots, n-1\}$.
We write

$$G_0 = \{g \in G : g(0) = 0\}.$$

G_0 is the subgroup of G that *stabilizes* the point 0.
The orbit of the element 0 under the action of G is

$$G(0) = \{g(0) : g \in G\} = \{x_{0,1}, x_{0,2}, \ldots, x_{0,n_0}\}.$$

We form $\mathcal{U}_0$ by choosing for each element $x_{0,i}$ in the orbit of 0an element $h_{0,i}$ in G, such that $h0, i(0) = x_{0,i}$.
Now $\mathcal{U}_0$ is a left transversal of G_0 ($G = U_0 G_0$): Every $g \in G$ maps 0 onto some $x_{0,i}$, $g = h_{0,i}\left(h_{0,i}^{-1} g\right)$, and $h_{0,i}^{-1} g \in G_0$. Thus $g \in h_{0,i} G_0$.
$\mathcal{U}$ only contains one representative from each coset of G_0: the elements in each coset $h_{0,i} G_0$ map 0 onto a different $x_{0,i}$.

Schreier-Sims

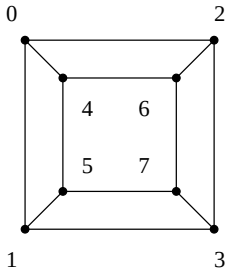
Let us apply the idea recursively:

$$\begin{aligned} G_0 &= \{g \in G : g(0) = 0\} \\ G_1 &= \{g \in G_0 : g(1) = 1\} \\ G_2 &= \{g \in G_1 : g(2) = 2\} \\ &\vdots \\ G_{n-1} &= \{g \in G_{n-2} : g(n-1) = n-1\} = \{\mathbf{I}\} \end{aligned}$$

Now $G \supseteq G_0 \supseteq \ldots \supseteq G_{n-1} = \{\mathbf{I}\}$.

The Schreier-Sims representation of the group $G$is $G = \mathcal{U}_0\mathcal{U}_1 \ldots \mathcal{U}_{n-1}$.

Schreier-Sims: Example



For the cube graph we may choose e.g.

$$\mathcal{U}_0 = \left\{ \begin{array}{l} (0)(1)(2)(3)(4)(5)(6)(7), \\ (0,1,3,7,6,4)(2,5), \\ (0,2,6,4)(1,3,7,5), \\ (0,3,6)(1,7,4)(2)(5), \\ (0,4,6,7,3,1)(2,5), \\ (0,5,3,6)(1,7,2,4), \\ (0,6,3)(1,4,7)(2)(5), \\ (0,7)(1,6)(2,5)(3,4) \end{array} \right\}$$

$$\mathcal{U}_1 = \left\{ \begin{array}{l} (0)(1)(2)(3)(4)(5)(6)(7), \\ (0)(1,2)(3)(4)(5,6)(7), \\ (0)(1,4,2)(3,5,6)(7) \end{array} \right\}$$

$$\mathcal{U}_2 = \left\{ \begin{array}{l} (0)(1)(2)(3)(4)(5)(6)(7), \\ (0)(1)(2,4)(3,5)(6)(7) \end{array} \right\}$$

and $\mathcal{U}_3, \ldots, \mathcal{U}_7 = \{(0)(1)(2)(3)(4)(5)(6)(7)\}$.

Schreier-Sims

For a group G, when we know the Schreier-Sims representation $\mathcal{U}_0\mathcal{U}_1 \ldots \mathcal{U}_{n-1}$, it is easy to go through all elements in a recursive fashion: just compute all $g = u_0u_1 \ldots u_{n-1}$, where $u_i \in \mathcal{U}_i$.

Testing whether a given g is in G goes as follows. Every $g \in G$ can be written in the form $u_0u_1 \ldots u_{n-1}$. First we examine $g(0)$ to deduce, which $u_0 \in \mathcal{U}_0$ must be chosen (the one for which $u_0(0) = g(0)$). After this the problem is reduced to testing whether $u_0^{-1}g \in G_0$, that is, we will try to write $u_0^{-1}g$ in the form $u_1 \ldots u_n$, etc.

```
Test(n, g, G = [U_0, ..., U_{n-1}]):
for i ← 0 to n − 1:
    if there is a h ∈ U_i, for which h(i) = g(i):
        g ← h^{-1} g
    else:
        return i
return n
```


Computing a Schreier-Sims representation

```
enter(n, g, G = [U_0, U_1, ..., U_{n-1}]):
i ←test(n, g, G = [U_0, U_1, ..., U_{n-1}])
if i = n
    return
U_i ← U_i ∪ {g}
for j = 0 to i:
    for h ∈ U_j:
        enter(n, gh, G)

main:
for i ← 0 to n − 1:
    U_i ← {I}
for α ∈ Γ:
    enter(n, α, G = [U_0, ..., U_{n-1}])
return G
```

The Enter function tests whether g belongs to the group G, given in Schreier-Sims form, and if not, it adds g to the generators.

Schreier-Sims basis change

Previously the points were fixed in the order $0, \ldots, n-1$. Of course the points may be fixed in an arbitrary order. We choose a permutation β of the elements $\{0, \ldots, n-1\}$, and

$$G_0 = \{g \in G : g(\beta(0)) = \beta(0)\},$$

and

$$G_i = \{g \in G_{i-1} : g(\beta(i)) = \beta(i)\}.$$

All operations are performed exactly analogously.

As a new operation, we have changing the basis: change the group given in β to the basis β'. This can be done by using the Enter procedure to add each of the permutations in basis β to the Schreier-Sims representation in basis β'.

Minimum representative of the orbit of a k-permutation

Suppose that we have a k-permutation $t = (t_1, \ldots, t_k)$, whose elements $t_i \in X$. When G acts on the ordered set X, it induces an action on the set of k-permutations. We will find the lexicographical minimum representative $\min G(t)$ of the orbit.

First, t_1 must be mapped to an element that is as early in the ordering of T as possible.We will compute $t_1' = \min G(t_1)$ e.g. by applying the generators of G and breadth-first search, and we also find a g for which $t_1' = g(t_1)$. We then compute $t' = g(t)$ and next we find $\min G_{t_1'}(t')$, etc. The necessary stabilizer-subgroups can be computed for example by Schreier-Sims basis changes.

Minimum representative of the orbit of a k-subset

Suppose that we have a k-subset $T = \{t_1, \ldots, t_k\}$ of an ordered set X. When G acts on X, it induces an action on the k-subsets. We will determine the lexicographical minimum representative $\min G(T)$ of the orbit.

For each t_i we find the minimum element of its orbit $\min G(t_i)$ and the corresponding group element g_i. Suppose that $t' = \min g_i(t_i)$ with g' the corresponding element. We compute $T' = g'(T)$ and apply the method recursively to determine $\min G_{t_1'}(T')$.

If at some stage there are several t_i, for which $t' = g_i(t_i)$, we must use backtracking search to consider each alternative in turn.

Again, the necessary stabilizer subgroups can be computed by Schreier-Sims basis changes.

Invariants

A function ϕ is a graph invariant, if its value does not depend on the labelling of the vertices:

$$\phi(G) = \phi(\pi(G)) \text{ for all } \pi \in \mathrm{Sym}(V).$$

For example when $\mathcal{V} = \{v_1, \ldots, v_n\}$,

$$\phi(G) = [\deg(v_1), \ldots, \deg(v_n)]$$

is not an invariant, but the multiset

$$\phi(G) = \{\deg(v_1), \ldots, \deg(v_n)\}$$

is; thus a graph invariant can be obtained by sorting the list of vertex degrees in ascending order. If $\phi(G_1) \neq \phi(G_2)$, then G_1 and G_2 cannot be isomorphic.

Vertex invariants

Let F be a family of graphs over the vertex set V. The function $D : F \times V \mapsto R$ is a vertex invariant, if its value does not depend on the labeling of the vertices:

$$D(G, v) = D(\pi(G), \pi(v)) \text{ for all } \pi \in \text{Sym}(V).$$

For example $\deg(v)$ or the number of triangles that contain v. For later use we assume that R is totally ordered.

Of invariants

We may use vertex invariants to construct graph invariants. For example the vertex invariant $D : F \times V \mapsto R$ gives us the graph invariant $\phi_{D,r}(G) = |B_D[r]|$, where $B_D[r] = \{v \in V : D(G, v) = r\}$.

Vertex and graph invariants can be combined to form new invariants:

$$\phi(G) = [\phi_1(G), \ldots, \phi_n(G)]$$

and

$$D(G, v) = [D_1(G, v), \ldots, D_n(G, v)].$$

The order of the values of $D(G, v)$ can be chosen to be e.g. the lexicographical order of lists.

Vertex invariants yield new vertex invariants, e.g., how many edges connect v to vertices in $B_D[r]$:

$$D'_r(G, v) = |\{\{v, v'\} \in E : v' \in B_D[r]\}|.$$

Certificates

Two nonisomorphic graphs may have the same invariant. For a family of graphs $\mathcal{F}$, a certificate c is a function for which $c(\mathcal{G}_1) = c(\mathcal{G}_2)$ if and only if $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{F}$ are isomorphic.

A certificate is also an invariant.

Eccentricity of a vertex and center of a tree

In a graph, let $d(v_1, v_2)$ be the length of the path between v_1 and v_2. Let $e(v) = \max_{v' \in V} d(v, v')$ be the eccentricity of v.

The center of a connected graph consists of the vertices with minimum eccentricity. The center of a tree contains at most 2 vertices, which are neighbors of each other.

Proof: Let $e(v_1) = e(v_2) \leq e(v')$ for all $v' \in V$, let $\{v_1, v_2\} \notin E$ and let v_3 some vertex on the path from v_1 to v_2. Let v_4 be a vertex for which $d(v_3, v_4) = e(v_3)$. Either the path from v_1 to v_4 or the path from v_2 to v_4 travels via v_3; thus either $e(v_1) > e(v_3)$ or $e(v_2) > e(v_3)$ — a contradiction. Since the vertices in the center are neighbors, they form a clique, but in a tree the maximum possible clique has two vertices.

If a tree contains internal nodes, a leaf node cannot be in the center, since its neighbor will have lower eccentricity. If we remove the leaf nodes from such a tree, the eccentricity of the remaining vertices is reduced by one.

A certificate for rooted trees

A rooted tree is a tree where one vertex has been designated as root. We compute a certificate: we remove the root v, after which we have one or more subtrees. We compute the certificate for each of the subtrees, with the neighbor of v as the root. The certificate is then obtained by concatenating 0, the certificates of the subtrees in lexicographical order, and 1.

A certificate for trees

If there is only one vertex in the center of the tree, use it as root and compute the certificate as for a rooted tree.

If there are two vertices in the center, remove the edge between them, and consider each of them as root for computing the certificate for the subtrees. Finally, concatenate the certificates in lexicographical order.

A certificate for trees

The certificate on the previous slide can be computed as follows. This method searches for the center while computing the certificate, and parts of the certificate may end in a slightly different order.

Label each vertex with 01.
As long as there are at least 2 vertices:

set $T \leftarrow$ internal nodes (deg > 1)

for each $x \in T$:

- from the label of x, remove the 0 at start and 1 at end
- form the multiset Y from the labels of x and its neighbors
- concatenate the elements of Y in lexicographical order, prepend a 0 and append a 1, and label x with the result

remove the neighboring leaf nodes from x

If only one vertex remains, its label is the certificate; if two vertices remain, the certificate is obtained by concatenating their labels in lexicographical order

A certificate for graphs

When permuting the vertices of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the permutation $\pi \in \mathrm{Sym}(\mathcal{V})$ we obtain the incidence matrix

$$A_\pi(\mathcal{G})[u,v] = \begin{cases} 1, \text{ if } \{\pi(u), \pi(v)\} \in \mathcal{E} \\ 0 \text{ otherwise} \end{cases}$$

$\mathrm{Num}_\pi(\mathcal{G})$ is obtained by reading the elements below the diagonal in $A_\pi(\mathcal{G})$ as a binary number:
$a_{21}\, a_{31} a_{32}\, a_{41} \ldots a_{43}\, a_{51} \ldots a_{54} \ldots a_{n1} \ldots a_{n\,n-1}$
In computing the simple certificate:

$$\min\{\mathrm{Num}_\pi(G) : \pi \in \mathrm{Sym}(V)\}$$

we simultaneously determine the maximum independent set, which is an **NP**-hard problem. However, graph isomorphism is not believed to be that difficult.

Idea: order the vertices in an order determined by some vertex invariants. Partition the vertices accordingly into an ordered partition B. Let Π_G be the set of permutations that preserve the ordered partition: if $u \in B_i$ and $v \in B_j$, then $\pi(u) < \pi(v)$, if $i < j$. Now

$$\mathrm{cert}(G) = \min\left\{\mathrm{Num}_\pi(G) : \pi \in \Pi_G\right\}.$$

Certificate for graphs / refining a partition

Let $B = [B_{r_0}, \ldots, B_{r_{k-1}}]$ be an ordered partition (based on vertex invariants) of the vertices of G. If B is discrete (each nonempty $B[i]$ contains exactly one element), we are done; otherwise we will try to form even better vertex invariants, so that Π_G would be reduced in size..

We write $D_T(\mathcal{G}, v) = |v' : \{v, v'\} \in \mathcal{E}, v' \in T|$. This invariant tells us the number of neighbors v has in T.

We will refine the partition: if there are vertices $u, v \in B_{r_i}$ and some $T = B_{r_j}$ such that $D_T(\mathcal{G}, u) \neq D_T(\mathcal{G}, v)$, we partition B_{r_i} into smaller parts according to D_T and order the new smaller partitions in ascending order of values of D_T. When $D_{B_{r_i}}(\mathcal{G}, u) = D_{B_{r_i}}(\mathcal{G}, v)$ for all i and $u, v \in B_i$, the partition is B is equitable. It is important that the order in which the refining operations are carried out is invariant!

For example:

Refine(A):
$B \leftarrow A$
let S be a list of elements of B
while $S \neq \emptyset$:
    remove the first element T from S
    for each $B[i] \in B$ (in order):
        for each h: $L[h] \leftarrow \{v \in B[i] : D_T(\mathcal{G}, v) = h\}$
        if there are more than one nonempty $L[h]$:
            replace $B[i]$ with the sets $L[h_1] \ldots L[h_n]$ (in order)
            append the sets $L[h]$ to S (in order)

A certificate for graphs

We will compute a certificate for the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. We start from the partition $B = \{B_0\}$, where $B_0 = \mathcal{V}$. We refine the partition until it is discrete, and then we will permute the vertices according to the discrete ordered partition, and find the value of the certificate.

If the partition is equitable but not discrete, we will find the first set with more than one element. For each element in that set in turn, we will split that element into a part of its own and apply recursively; the certificate is then the minimum value obtained in any search branch.

cert$(B, \mathcal{G})$:
refine(B)
if B is discrete:
    compute π from B; return Num$_\pi(\mathcal{G})$
else:
    find the least i, for which $|B_i| > 1$
    $best \leftarrow \infty$
    for each $x \in B_i$:
        $B' = [B_0, \ldots, B_{i-1}, \{x\}, B_i \setminus \{x\}, B_{i+1}, \ldots]$
        $t \leftarrow$ cert$(B', \mathcal{G})$
        if $t < best$: $best \leftarrow t$
    return $best$

Using symmetries

If we obtain the same certificate value in two search branches, $\mathrm{Num}_\pi(G) = \mathrm{Num}_\mu(G)$, then $\pi(G) = \mu(G)$, and $\pi^{-1}\mu(G) = G$, so $\pi^{-1}\mu$ is an automorphism of G.

We may consider the automorphisms found as generators of a group and present them in the Schreier-Sims form.

When we have reached the point in the search where B_i is the first set with $|B_i| > 1$, we first choose some $x \in B_i$ and examine that branch as before. After this we can perform a basis change with the known automorphisms such that for $k < i$ β_k is the element in B_k, and $\beta_i = x$. Now $\mathcal{U}_i(x)$ is the orbit of x under the known automorphisms that stabilize B_1 to B_k; from that orbit it suffices to consider x only.

Naturally, if (at least a part of) the automorphism group is known in advance, we may enter that into the Schreier-Sims representation in advance.

Isomorphism of set systems

The isomorphism of set systems $(\mathcal{X}, \mathcal{B})$ can be treated as graph isomorphism as follows:

Represent the set system as a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{X} \cup \mathcal{B}$, and $\mathcal{E} = \{\{x, B\} : x \in \mathcal{X}, B \in \mathcal{B}, x \in B\}$. After this we only need to take care that we will not confuse the $\mathcal{X}$ vertices and $\mathcal{B}$ vertices; we can initialize the certificate computation with the vertex partition $[\mathcal{X}, \mathcal{B}]$.

Subset orbits

Let G be a permutation group on $\mathcal{X}$ and $S \subseteq \mathcal{X}$. The induced action of a group element $g \in G$ on S is such that $g(S) = \{g(s) : s \in S\}$. Thus G also permutes the subsets of $\mathcal{X}$.

The orbit of S is $G(S) = \{g(S) : g \in G\}$. If a set system has a nontrivial automorphism group, the set of its blocks must be a union of the subset orbits: if $S \in B$, we must also have $g(S) \in B$ for all $g \in G$.

The stabilizer of S in G is $G_S = \{g \in G : g(S) = S\}$. Again, G_S is a subgroup of G, as it is nonempty and closed.

Lemma: $|G| = |G(S)| \cdot |G_S|$.

Proof: As on the slide "The orbit of an element"; the group is thought to act on the subsets of $\mathcal{X}$.

There are $|G| / |G_S|$ left cosets, and each of them maps S onto different sets, so $|G(S)| = |G| / |G_S|$.

Subset orbits. Example: Ramsey number

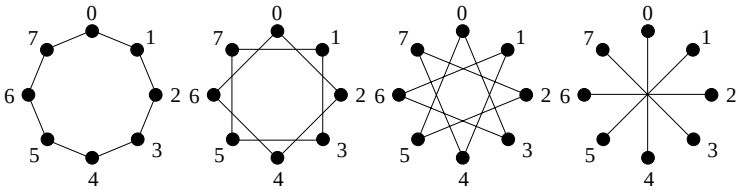
The Ramsey number $R(k, l)$ is the least integer n, for which all n-vertex graphs contain a k-vertex clique or an l-vertex independent set. We show that $R(3, 4) > 8$ by finding an 8-vertex graph with no 3-vertex clique and no 4-vertex independent set.

We shall limit the search space by guessing that we may find a graph $G = (\mathcal{V} = \{0, 1, \ldots, 7\}, \mathcal{E})$ whose automorphism group contains the cyclic group: $\langle(0, 1, 2, 3, 4, 5, 6, 7)\rangle$. That is, we require $\mathcal{E}$ to be a union of orbits of 2-subsets of $\mathcal{V}$ under $\langle(0, 1, 2, 3, 4, 5, 6, 7)\rangle$.

Subset orbits. Example: Ramsey number

The orbits of 2-subsets of $\mathcal{V}$ are

$$
\begin{aligned}
O_1 &= \{\{0,1\},\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{5,6\},\{6,7\},\{0,7\}\} \\
O_2 &= \{\{0,2\},\{1,3\},\{2,4\},\{3,5\},\{4,6\},\{5,7\},\{0,6\},\{1,7\}\} \\
O_3 &= \{\{0,3\},\{1,4\},\{2,5\},\{3,6\},\{4,7\},\{0,5\},\{1,6\},\{2,7\}\} \\
O_4 &= \{\{0,4\},\{1,5\},\{2,6\},\{3,7\}\}.
\end{aligned}
$$



By trial and error we may find that the edge sets $\mathcal{E} = O_3 \cup O_4$ ja $\mathcal{E} = O_1 \cup O_4$ satisfy our criteria.

Subset orbits. Example: Ramsey number $R(5,9) > 120$

There is a 120-vertex graph with no 5-vertex clique and no 9-vertex independent set. It can be found by a tabu search:

- Partition the edges into orbits under $G = \langle (1,\ldots,120) \rangle$.
- Choose a random subset of the orbits
- Repeatedly add or remove the edges in such an orbit that the change moves us to a graph with as few 5-vertex cliques and 9-vertex independent sets as possible.
- However, never add or remove edges in an orbit that has been added or removed within the previous 12 moves.

$(\mathcal{V},\mathcal{E})$, where $\mathcal{E} = \{\{v, v+d \pmod{120}\} : d \in S, v \in \mathcal{V}\}$, $\mathcal{V} = \{0,\ldots,119\}$ ja $S = \{2,3,6,7,13,15,17,18,19,20,22,23,28,29,31,33,41,42,43,45,48,52,53,54,60\}$, satisfies the conditions.

Generating symmetrical objects

If the search space for a combinatorial object is too large, we may limit the search space by limiting the search to objects with (at least) a given automorphism group.

Example

$\mathcal{X} = \{0,\ldots,24\}$ ja $G = \langle (0,1,\ldots,24) \rangle$.

When $\mathcal{B}$ is the union of the orbits of the sets $\{0,8,13\}$, $\{0,2,3\}$, $\{0,4,11\}$ and $\{0,6,15\}$, then $(\mathcal{X},\mathcal{B})$ is $\mathrm{STS}(25)$. (A Steiner triple system with $|\mathcal{X}| = 25$; each pair in $\mathcal{X}$ appears in exactly one triple in $\mathcal{B}$.

We could of course list all hundred triples.

Orbit incidence matrices

When G is a permutation group on $\mathcal{X}$ and $0 \le t \le k \le |\mathcal{X}|$, the orbit incidence matrix A_{tk} is an $N_t \times N_k$-matrix, where row i corresponds to the t-subset orbit Δ_i, column j corresponds to the k-subset orbit Γ_k, and $a_{ij} = \left|\left\{K \in \Gamma_j\right\} : K \supset T_0\right|$, where $T_0 \in \Delta_i$.

It turns out that $a_{ij} = \left|\left\{K \in \Gamma_j\right\} : K \supset T_0\right|$ does not depend on the chosen $T_0 \in \Delta$:

If $T_0, T_0' \in \Delta$, there is some $g \in G$ for which $g(T_0) = T_0'$. If $T_0 \subseteq K \in \Gamma$, then $T_0' \subseteq g(K)$.

Orbit incidence matrix. Example.

Let $\mathcal{X} = \{0,\ldots,4\}$ and $G = \langle(0,1,2,3,4)\rangle$.
The 2-subset orbits are

$$\begin{aligned}\Delta_1 &= \{\{0,1\},\{1,2\},\{2,3\},\{3,4\},\{4,0\}\} \text{ and}\\ \Delta_2 &= \{\{0,2\},\{1,3\},\{2,4\},\{3,0\},\{4,1\}\}.\end{aligned}$$

The 3-subset orbits are

$$\begin{aligned}\Gamma_1 &= \{\{0,1,2\},\{1,2,3\},\{2,3,4\},\{3,4,0\},\{4,0,1\}\} \text{ and}\\ \Gamma_2 &= \{\{0,1,3\},\{1,2,4\},\{2,3,0\},\{3,4,1\},\{4,0,2\}\}.\end{aligned}$$

The orbit incidence matrix $A_{23} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$. For example a_{22} can be computed by choosing $T_0 = \{0,2\} \in \Delta_2$ and observing that T_0 is contained in two of the sets in Γ_2, that is, in ($\{2,3,0\}$ and $\{4,0,2\}$).

Computing the orbit incidence matrix

The following algorithm computes the orbit incidence matrix in a naive manner. R and S are the sets of orbit representatives of t- and k-subsets ($t \le k$) respectively.

```
for g ∈ G:
    for T ∈ R:
        for K ∈ S:
            if T ⊆ g(K):
                A[T,K] ← A[T,K] + 1
for K ∈ R:
    stab ← 0
    for g ∈ G:
        if g(K) = K:
            stab ← stab + 1
    for T ∈ R:
        A[T,K] ← A[T,K]/stab
```


Burnside's lemma

(Frobenius, 1887) If a finite group G acts on a finite set X, and N is the number of orbits, then

$$N = \frac{1}{|G|}\sum_{g\in G} F(g),$$

where $F(g)$ is the number of $x \in \mathcal{X}$ for which $gx = x$.

Proof: in the above sum each $x \in \mathcal{X}$ is counted $|G_x|$ times (by definition of G_x). If x and y are in the same orbit, then $|G_x| = \left|G_y\right|$, so every one of the $|G|/|G_x|$ elements is counted $|G_x|$ times; in total, $|G|$ times. Each orbit contributes $|G|$ to the sum, so dividing the sum by $|G|$ gives us the number of orbits.

Burnside's lemma on k-subsets

Let us consider k-subsets of some set, upon which a permutation group acts in the natural way.

We denote with $F(g)$ the number of k-subsets fixed by g:

$$F(g) = \left|\{S \subseteq \mathcal{X} : |S| = k \text{ ja } g(S) = S\}\right|.$$

To compute $F(g)$ we first find the lengths of the cycles in g and write

$$\text{type}(g) = [t_1,\ldots,t_n],$$

where t_i is the number of cycles of length i. If $g(S) = S$, then S is the union of the elements in some cycles of g.

If S contains c_i cycles of i vertices, then we must have $c_i \le t_i$, and $k = \sum_i i c_i$. For given values of c_i there are $\prod_i \binom{t_i}{c_i}$ such sets.

Burnside's lemma on k-subsets

For given k and $[t_1, \ldots, t_n]$ we compute all possible combinations of c_i for which $c_i \leq t_i$ and $k = \sum_i i c_i$:

chiG(n, k, i, t):
if $i = 1$: $\chi \leftarrow 0$
if $i = n + 1$:
    if $k = 0$:
        $\chi \leftarrow \chi + \prod_i \binom{t_i}{c_i}$
    return
$C_i \leftarrow \{0, \ldots, \min(t_i, \lfloor k/i \rfloor)\}$
for $x \in C_i$:
    $c_i \leftarrow x$
    chiG$(n, k - ic_i, i + 1, t)$
return χ

Burnside's lemma. Example

How many essentially different flags with five stripes are there, when each stripe is either blue, white, or red? Flags are not considered essentially different, if one is obtained from the other by mirroring.

Flags may be viewed as lists $[c_1, c_2, \ldots, c_n]$, where each $c_i \in C$. There are a total of $|C|^n$ color combinations. The group consists of two permutations, identity and the mirroring τ, which acts on the flag such that $\tau\,[c_1, \ldots, c_n] = [c_n, \ldots, c_1]$. For each of these permutations π we compute number of flags fixed by the permutation. $F(\mathbf{I}) = |C|^n$ and $F(\tau) = |C|^{\lceil n/2 \rceil}$ so the number of flags is
$N = \frac{1}{2}\left(|C|^n + |C|^{\lceil n/2 \rceil}\right)$ or in this example
$\frac{1}{2}(243 + 27) = 135$.