T-79.5202 Combinatorial algorithms

Harri Haanpää

Laboratory for Theoretical Computer Science, TKK

Spring 2007

T-79.5202 Combinatorial algorithms

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

Combinatorial structures

A list: an ordered collection of elements, e.g. $X = [0, 1, 3, 0]$

A set: an unordered collection of elements without repetition, e.g. $X = \{1, 3, 4\}$. $|X|$ is the number of elements in X. The Cartesian product $X \times Y = \{[x, y] \,|\, x \in X \wedge y \in Y\}$.

Subset: Set X is a subset of set Y, if for all $x \in X$ also $x \in Y$. If $|X| = k$, then X is a k-subset of Y.

A graph: $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the set of vertices and edges, respectively. Each edge is a set of two vertices.

Example: Latin squares

A set system: $(\mathcal{X}, \mathcal{B})$, where $\mathcal{X}$ is a finite set and $\mathcal{B}$ a set of subsets of $\mathcal{X}$. (e.g. a partition of $\mathcal{X}$)

A Latin square: $n \times n$ array, each of whose rows and columns contains each of the numbers $\mathcal{Y} = \{1, \ldots, n\}$ exactly once, e.g.

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

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

A transversal design $TD_\lambda(k, n)$: $(\mathcal{X}, \mathcal{B})$, where
$|\mathcal{X}| = kn$; $\mathcal{X} = \mathcal{X}_1 \cup \ldots \cup \mathcal{X}_k$; $|\mathcal{X}_i| = n$;
$|B \cap \mathcal{X}_i| = 1$ for all $B \in \mathcal{B}$ and $1 \leq i \leq 3$;
for all $x \in \mathcal{X}_i$, $y \in \mathcal{X}_j$, $i \neq j$ there are λ blocks $B \in \mathcal{B}$, for which $\{x, y\} \subset B$.

Problem types

- enumerate combinatorial structures of a given kind
  - enumerate the possible poker hands
- determine the number of combinatorial structures of a given kind
  - find how many n bit binary words without two consecutive ones are there
- find a combinatorial structure of a given kind
  - color the vertices of a graph with 3 colors so that the endpoints of each edge are colored with different colors

Variants of a search problem

The knapsack problem: Given n items with weights $w_1, \ldots w_n$ and profits $p_1, \ldots, p_n$. A subset $S \subseteq \{1, \ldots, n\}$ fits into the knapsack, if $\sum_{i \in S} w_i \leq M$, where M is the capacity of the knapsack. Then the total profit is $P(S) = \sum_{i \in S} p_i$.

1. Is it possible to find some S, for which $P(S) = P$? (An NP-complete decision problem!)
2. Determine some S for which $P(S) = P$.
3. What is the maximum $P(S)$ obtainable?
4. Which S yields the maximum $P(S)$?

Solution strategies

A greedy algorithm: build the solution by making at each step the choice that appears best at short sight. E.g. for the knapsack problem put items into the bag in order of decreasing profit/weight-ratio until the knapsack is full.

Dynamic programming: When parts of the optimal solution are optimal solutions of the corresponding subproblems, we can start from solving the smallest subproblems first and use solutions to small subproblems to construct solutions of larger and larger subproblems.

Divide and conquer: Split the problem into subproblems, solve them and combine the solutions.

Backtrack search: Try out recursively all possible solutions.

Local search: Try to find a good solution by starting from an arbitrary solution and making a large number of small improvements.

Data structures for subsets

Size of set? Necessary operations? Insertion/deletion, testing membership, union, intersection, number of elements, listing elements?

- an (ordered) linked list of elements
  - when the base set is large and the subset is small
- binary trees
  - when the base set is large and the subset is smallish
- bit map representation
  - when the base set is small

e.g. $S = \{1, 3, 11, 16\} \subset \{0, \ldots, 16\}$ can be expressed as the bit string 01010000000100001, which can be split into e.g. 8-bit words into an array:
$A[0] = 01010000_2$, $A[1] = 00010000_2$,
$A[2] = 10000000_2$

Data structures for graphs and set systems

1. Set of edges
2. Incidence matrix: a matrix whose rows and edges correspond to the nodes and edges of the graph; a matrix entry is 1, if the corresponding node is an endpoint of the corresponding edge, or 0 otherwise
3. Adjacency matrix: a matrix whose rows and edges correspond to vertices; an element is 1, if the two corresponding vertices are connected by an edge
4. Adjacency list: For each vertex, list the neighboring vertices

1. and 2. can also be used for set systems

rank and unrank functions

Order the $\binom{39}{7}$ lottery tickets. At which position does 3, 8, 12, 14, 15, 32, 38 appear? Which ticket is in position 3937483?

Let S be a set of some combinatorial structures. Let us enumerate them $0 \ldots |S| - 1$:

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

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

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

- a random $s = \text{unrank}(\text{random}(0 \ldots |S| - 1))$
- an integer representation is compact

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

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

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

The structures can be enumerated with the successor function starting from $\text{unrank}(0)$.

Lexicographical order of lists

Let us order lists $l = [s_1, s_2, \ldots, s_n]$ and $l' = [s'_1, s'_2, \ldots, s'_{n'}]$ as follows: If one list is a prefix of the other one, the shorter list precedes the longer one. Otherwise find the least i, for which $s_i \neq s'_i$. If $s_i \prec s'_i$, then $l \prec l'$, and vice versa.

E.g. List of 3 letters; for ['A','B','C'] we write here 'ABC'.

Let S = {'A','B','C',...,'Z'}. Order the alphabet as usual: $A \prec B$, etc.

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

Now the order is 'AAA'$\prec$'AAB'$\prec \ldots \prec$'ZZY'$\prec$'ZZZ', and in this special case we have
$\text{rank}([s_1 s_2 s_3]) = |S|^2 \text{rank}(s_1) + |S| \text{rank}(s_2) + \text{rank}(s_3)$. (cf. 26-ary numbers)

Lexicographical order of subsets

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

Consider the subsets of $S = \{1,\ldots,n\}$.
When $T \subseteq S$, the characteristic vector of T is $\chi(T) = [x_{n-1},\ldots,x_0]$, where $x_i = 1$, if $n-i \in T$, and $x_i = 0$, if $n-i \notin T$. (cf. bit map representation)
Order the subsets according to the lexicographical order of their characteristic vectors.

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

Lexicographical rank of a subset

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

Let $V = \{1,\ldots,8\}$.
E.g. 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

Lexicographical unrank of a subset

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

Let $V = \{1,\ldots,8\}$.
E.g. 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}

If the base set is not $\{1,\ldots,n\}$ (e.g. $\{0,\ldots,n-1\}$), it may be useful to map the base set (bijectively) onto $\{1,\ldots,n\}$.

Minimum change ordering

Occasionally desiderable: two consecutive structures differ as little as possible. For subsets distance can be e.g.

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

E.g. the distance of the sets subsetlexunrank $(n,3) = \{2,3\}$ and subsetlexunrank $(n,4) = \{1\}$ is 3, when $n = 3$.

For subsets there exists orderings, where the dist of consecutive sets is always 1. The characteristic vectors of such an ordering form a Gray code.

Gray codes

A Gray code is a list of 2^n n-bit binary words, where each n-bit binary word appears exactly once, and the Hamming distance of consecutive words is 1. (Sometimes the Hamming distance between the first and last codeword is also required be 1.)

A nice family of Gray codes (binary reflected Gray codes): $G_1 = [0, 1]$, G_{i+1} is obtained from G_i by taking two copies of it, prepending 0 to each codeword in the first copy and 1 to each codeword in the second, reversing the order of the codewords in the second copy and concatenating the result:

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

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

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

Let the binary representation of a codeword be $a_{n-1} \ldots a_0$ and the binary representation of its rank number be $b_{n-1} \ldots b_0$.

$a_j = \left(b_j + b_{j+1}\right) \mod 2$ and $b_j = \sum_{i=j}^{n-1} a_i \mod 2$.

Lexicographical order of k-element subsets

$S = \{1, \ldots, n\}$. Generate all $\binom{n}{k}$ subsets with k elements.
Represent $T \subseteq S$ as a list: $\vec{T} = [t_1, \ldots t_k]$, $t_i < t_{i+1}$, and order the subsets by the lexicographical order of these lists.

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

Successor: increment the largest element that can be incremented, and set elements larger than it to be as small as possible.

rank $(T) = \sum_{i=1}^{k} \sum_{j=t_{i-1}+1}^{t_i - 1} \binom{n-j}{k-i}$, where $t_0 = 0$.

Co-lex order of k-element subsets

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

$S = \{1, \ldots, n\}$. Enumerate all $\binom{n}{k}$ subsets with k elements.
Present $T \subseteq S$ as a list: $\overleftarrow{T} = [t_1, \ldots t_k]$, $t_i > t_{i+1}$, and order the subsets by the lexicographical order of these lists.
Successor: increment the least element that can be incremented, and set elements less than that to their least possible values.
rank $(T) = \sum_{i=1}^{k} \binom{t_i - 1}{k+1-i}$
rank is independent of n !

Connection between lex and co-lex order

Map each set $T \subseteq \{1,\dots n\}$ to $T' = \{n+1-t | t \in T\}$. The lex order of the sets T is the reverse co-lex order of the sets T', and vice versa!

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

Using this transformation makes it easier to compute the lexicographical rank and unrank of k-subsets.

Example: rank of a k-subset

Sort the $\binom{39}{7}$ lottery tickets in lexicographical order. In what position does 3, 8, 12, 14, 15, 32, 38 appear?

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

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

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

Example: unrank of a k-subset

Order the $\binom{39}{7}$ lottery tickets lexicographically. Which ticket is in position 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	34	40414
5	40414	24	6765
4	6765	22	780
3	780	18	100
2	100	15	9
1	9	10	0

$$T' = \{38,34,24,22,18,15,10\},$$
$$T = \{2,6,16,18,22,25,30\}$$

Permutations

A permutation is a way of ordering the elements $\{1,\dots,n\}$, that is, a bijection from $\{1,\dots,n\}$ onto itself.

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

E.g.

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

A permutation can be represented as a list:
$[\pi(1), \pi(2), \dots, \pi(n)]$

E.g. $[3,5,1,4,6,2]$.

A permutation can be presented in cycle notation, where within each pair of parenthesis each element maps onto the next one and the last one onto the first, e.g.

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

Combining permutations

Permutations are functions and they are combined like functions: right to left.

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

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

Parity of permutations

The simplest permutation is the transposition of two elements (i, j), where $i \neq j$. Permutations may be divided into two classes:

Even permutations can only be expressed as the product of an even number of transpositions, e.g.

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

Odd permutations can only be expressed as the product of an odd number of transpositions, e.g.

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

Auxiliary result

If the lists $d = [d_1, \ldots, d_n]$, where $0 \leq d_i < n_i$ are ordered lexicographically, then

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

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
    $d_j = 0$

Lexicographical rank of permutations

We order permutations by the lexicographical order of their list presentations. E.g.

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

When choosing the ith element of the list there are $n + 1 - i$ remaining elements. We use d_i to denote the number of remaining elements that were smaller than the one we chose. Now$0 \leq d_i < n_i = n + 1 - i$, and

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

E.g. $\pi = [2,4,1,3] \Rightarrow d = [1,2,0,0]$ and

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

Lexicographical unrank and successor of permutations

Conversely unrank (10) first yields $d = [1,2,0,0]$, from which we obtain $\pi = [2,4,1,3]$.

Successor: We try not to disturb the elements at the beginning of the list; we find the shortest suffix of the list that is not in inverted lex. order. Within the suffix, we replace the first element of the suffix by the next larger element, and sort the remaining elements in ascending order. E.g.

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

Minimal change ordering of permutations: Trotter–Johnson

A minimal change for permutations is the transposition of two adjacent elements: $[\ldots,i,j,\ldots] \to [\ldots,j,i,\ldots]$.

Trotter (1962): starting with the minimal change order T^{n-1} of the elements $\{1,\ldots,n-1\}$, we add element n as follows. We make n copies of each permutation in T^{n-1}, and add n to these permutations at suitable places so that they form a zig-zag pattern.

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

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

This method was known already in 15th century England; in campanology (ringing church bells) this was known as 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):
$\pi' \leftarrow \pi$ without element n
$r \leftarrow$ TJRank $(\pi', n-1)$
if r is even:
    $r \leftarrow nr +$ no. elems. right of n :
else:
    $r \leftarrow nr +$ no. elems. left of n :
return r

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

Trotter–Johnson unrank

TJUnrank(r, n):
$\pi' \leftarrow$ TJUnrank $\left(\left\lfloor \frac{r}{n} \right\rfloor, n-1\right)$
$r \leftarrow r \bmod n$
if π' even:
    $\pi \leftarrow \pi'$, to which we add n s.t. r elems. remain to its right
else:
    $\pi \leftarrow \pi'$, to which we add n s.t. r elems. remain to its left
return π

E.g. TJUnrank$(13, 4)$:
    TJUnrank$(3, 3)$:
        TJUnrank$(1, 2) = [2, 1]$
    1 odd: insert 3 s.t. $3 \bmod 3 = 0$
elems. remain to its left: $[3, 2, 1]$
3 odd: insert 4 s.t. $13 \bmod 4 = 1$ elems. remain to its left: $[3, 4, 2, 1]$

Trotter–Johnson successor

TJSuccessor(π, n):
$\pi' \leftarrow \pi$ without element n
if π' is even and n can be moved left, do so
else if π' is odd and n can be moved right, do so
else compute TJSuccessor$(\pi', n-1)$ while keeping n in its place.

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

$\pi' = [3, 1, 2]$ is even, but 4 cannot be moved left; compute $[4]$ +TJSuccessor$([3, 1, 2])$:

$\pi' = [1, 2]$ is even, but 3 cannot be moved left; compute $[3]$ +TJSuccessor$([1, 2])$:

$\pi' = [1]$ is even, and 2 can be moved left: $[2, 1]$

we obtain $[4] + [3] + [2, 1] = [4, 3, 2, 1]$

Myrvold & Ruskey: unrank

Instead of choosing an order and finding rank and unrank functions for it, Myrvold and Ruskey chose a fast unrank function and designed the corresponding rank function.

The traditional method of constructing a random permutation:

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

where $r_i = \text{random}(1, \ldots, i)$. We obtain the permutation

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

(For simplicity, $(i, i) = (i)$ here.) Thus we can represent every permutation as the list $[r_n, r_{n-1}, \ldots, r_1]$ and sort the lists lexicographically. Every permutation has a unique representation of this form.

Unranking is simple: find the values of the r_i from the rank and use the above algorithm to construct the permutation.

Myrvold & Ruskey: rank

To obtain the rank we must first obtain the r_i and then compute

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

When π is presented in the form of the previous slide, only the leftmost transposition moves the element n, so $\pi(n) = r_n$. So from π we easily obtain r_n. Then we compute $(n, r_n)\pi$, which maps n onto itself, so we essentially have a permutation of $\{1, \ldots, n-1\}$, and we iterate.

The order is not particularly intuitive:

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

Integer partitions

$P(m)$: in how many ways can the positive integer m be expressed as a sum of positive integers $m = a_1 + \ldots + a_n$, when the order of the summands is not considered significant? (or equivalently $a_1 \geq \ldots \geq a_n$)

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

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

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

Generating partitions

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

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

The parameter m is the integer to be partitioned, B is the largest integer that can be chosen as the next a_i without violating the order, and L is a list of the sizes of the parts.

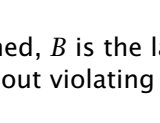
Ferrers-Young diagrams

The Ferrers-Young diagram is obtained by writing dots in lines, a_i dots in line i.

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

By transposing rows to columns we obtain the conjugate diagram and conjugate partition:

$D^* =$



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

$P(m, n)$:
there are as many partitions of m with n parts as there are partitions of m where n is the largest part.

Relation I

Clearly $P(m,m) = P(m,1) = 1$, when $m > 1$. We define $P(m,0) = 0$, when $m > 0$, and $P(0,0) = 1$.

Theorem 3.2: When $m \geq n > 0$,

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

Proof. Let $\mathcal{P}(m,n)$ denote the set of n-partitions of m. We partition $\mathcal{P}(m,n)$ into two sets and define the bijections:

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

Φ_1 and Φ_2 are bijections from parts of $\mathcal{P}(m,n)$ onto $\mathcal{P}(m-1,n-1)$ and $\mathcal{P}(m-n,n)$, so the sets contain the same number of elements.

Relations II

Theorem 3.3: When $m \geq n > 0$,

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

Proof: Split $\mathcal{P}(m,n)$ into parts $\mathcal{P}(m,n)_i$, each of which contains the partitions with exactly i parts greater than 1.

For each $0 \leq i \leq n$ define the bijection

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

as follows:

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

Rank function for $\mathcal{P}(m,n)$

Order the partitions $[a_1,\ldots,a_n]$ by the lexicographical order of their inverse standard form $[a_n,\ldots,a_1]$. E.g. $\mathcal{P}(10,4)$:

std. form	inv. std. form
$[7,1,1,1]$	$[1,1,1,7]$
$[6,2,1,1]$	$[1,1,2,6]$
$[5,3,1,1]$	$[1,1,3,5]$
$[4,4,1,1]$	$[1,1,4,4]$
$[5,2,2,1]$	$[1,2,2,5]$
$[4,3,2,1]$	$[1,2,3,4]$
$[3,3,3,1]$	$[1,3,3,3]$
$[4,2,2,2]$	$[2,2,2,4]$
$[3,3,2,2]$	$[2,2,3,3]$

Partitions with $a_n = 1$ precede those with $a_n > 1$. We obtain

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

Successor in $\mathcal{P}(m,n)$

Here exceptionally the elements in the lists are in ascending order, $a_i \leq a_{i+1}$.

The partition $[a_1,\ldots,a_n]$ is the last one, when $a_1 + 1 \geq a_n$. Then m is divided by n as equally as possible.

In finding the successor in lexicographical order we try to keep the beginning of the list unchanged.

Successor:

1. Find the shortest suffix of the list that is not equally partitioned, i.e., the greatest i, for which $a_i + 1 < a_n$.
2. Increment a_i by one and set $a_{i+1},\ldots,a_{n-1}$ to their minimum value ($= a_i$)
3. Justify the sum by setting $a_n = m - \sum_{i=1}^{n-1} a_i$.

E.g.:
with $[1,2,4,5,5]$ we find $i = 2$. Set $a_2 = a_2 + 1 = 3$, $a_3 = 3$, $a_4 = 3$ and $a_5 = 17 - 3 - 3 - 3 - 1 = 7$ to obtain $[1,3,3,3,7]$.

Labeled trees

A graph $G = (V, E)$ is a tree if its connected and cycle-free. The degree of a vertex v is the number of edges with v as one endpoint. Let $V = \{1, 2, \ldots, n\}$. There are then n^{n-2} different trees with the vertex set V.

Let T_n be the set of trees with vertex set V. Prüfer correspondence:

$$\text{Prüfer} : T_n \mapsto V^{n-2}$$

$$\text{Prüfer}^{-1} : V^{n-2} \mapsto T_n$$

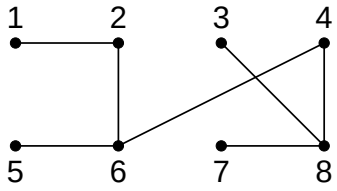
Prüfer

Prüfer(n, E):
for $i = 1$ to $n - 2$:
    let v be the highest-numbered vertex of degree 1
    find the edge $\{v, v'\} \in E$ and set $L_i \leftarrow v'$
    remove the edge $\{v, v'\}$

Each vertex v appears $\deg(v) - 1$ times in L. At the end only the edge $\{1, v\}$ remains; the degree of v is 1 when the algorithm terminates.

InvPrüfer(n, L):
compute the degrees of the vertices from L
append 1 to the list: $L_{n-1} \leftarrow 1$
for $i = 1$ to $n - 1$:
    let v be the highest-numbered vertex of degree 1
    add the edge $\{v, L_i\}$ to the graph
    decrement by one the degrees of v and L_i

Prüfer – example

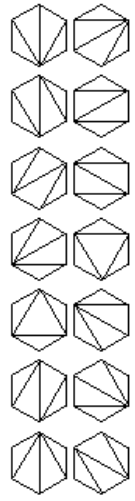


Vertex degrees	L_i	Edge
[1,2,1,2,1,3,1,3]	8	{7,8}
[1,2,1,2,1,3,0,2]	6	{5,6}
[1,2,1,2,0,2,0,2]	8	{3,8}
[1,2,0,2,0,2,0,1]	4	{4,8}
[1,2,0,1,0,2,0,0]	6	{4,6}
[1,2,0,0,0,1,0,0]	2	{2,6}
[1,1,0,0,0,0,0,0]		

From the list representation we easily obtain rank and unrank functions by interpreting the list as a number in base n.

Catalan numbers

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

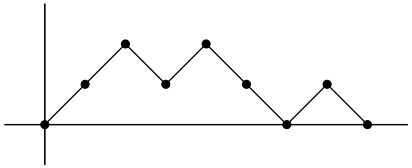


Catalan numbers appear in many contexts:

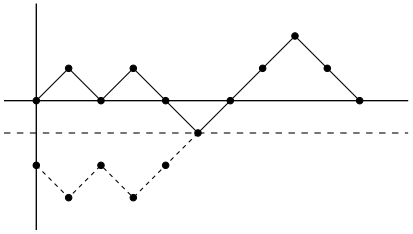
- how many ways are there to compute a matrix product expression so that two matrices are multiplied at a time: $((M_1\,(M_2 M_3))\,(M_4 M_5))$
- how many ways are there to triangulate an $n+2$-gon
- how many strings of $2n$ bits with n ones exist, where to the left of any position there are at least as many zeroes as ones: 000111, 001011, 001101, 010011, 010101

About Catalan numbers

Such binary strings can be represented as a mountain range that never goes below the 0 level. E.g. $a = 00101101$ corresponds to



We mirror those mountain ranges that go below 0 over the axis $y = -1$ from the beginning until they first go below 0. We obtain a bijection between mountain ranges that go below 0 and mountain ranges that go from $(-2, 0)$ to $(2n, 0)$.



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

Catalan rank and unrank

We compute how many ways of finishing the mountain range there are starting from each position, e.g. for 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: follow the mountain range; when it goes down and right, we add to rank the number that was up and right.

unrank: if rank $\geq$ the number up and right of the current position, go down and right and subtract the number that was up and right from rank, otherwise go up and right.

E.g. rank $(0010110101) = 22$

Backtrack search

- a common method for solving a combinatorial search, optimization or enumeration problem
- recursive: typically implemented by subroutines that call themselves while building solutions step by step
- complete search: the entire search space is examined
- pruning may spare us from having to look at inessential parts of the search space

Example of backtrack search

Knapsack problem: Given n with weights $w_1, \ldots w_n$ and profits $p_1, \ldots, p_n$. The capacity of the knapsack is M. Maximize $P(x) = \sum p_i x_i$ subject to $x_i \in \{0, 1\}$ and $\sum w_i x_i \leq M$.

We construct the list $[x_0, \ldots, x_{n-1}]$ recursively. Here len(x) is the length of list x, that is, the number of already fixed x_i; the recursion is started with 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])
```

Backtrack search in general

In many combinatorial problems, solutions can be presented as a list $X = [x_0, \ldots, x_{n-1}]$, where $x_i \in P_i$, where P_i is a finite set of possible values for x_i. A naive backtrack search constructs all elements in $P_0 \times P_1 \times \ldots \times P_{n-1}$. During the search the length of the list corresponds to the depth of the node in the search tree.

A partial solution $[x_0, \ldots, x_{l-1}]$ may limit the search; sometimes we can deduce that some $x_l \in P_l$ cannot lead to feasible solutions. Then we can prune the search and only consider the choice set $C_l \subseteq P_l$.

Backtrack(x):
if $x = [x_0, \ldots, x_{l-1}]$ is a feasible solution:
    process it
compute C_l
for each $c \in C_l$:
    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($x + [c]$)

Generating cliques

A clique in a graph $G = (\mathcal{V}, \mathcal{E})$ is such a subset $S \subseteq \mathcal{V}$ of the vertex set V that between all pairs of nodes $x, y \in S$, $x \neq y$ there is an edge: $\{x, y\} \in \mathcal{E}$.

A maximal clique is a clique that is not a subset of a larger clique.

Define the backtrack search:
$[x_0, \ldots, x_{l-1}]$ corresponds to the clique $S_l = \{x_0, \ldots, x_{l-1}\}$

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

Problem: the algorithm generates each k-vertex clique $k!$ times, once in each possible order! Solution: order the vertices $v_0 < \ldots < v_{n-1}$ and choose

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

Generating cliques II

First precompute for each vertex v the auxiliary sets $N_v = \{u \in \mathcal{V} : \{u, v\} \in E\}$ and $G_v = \{u \in \mathcal{V} : u > v\}$. N_v is the set of neighbors of v and G_v is the set of vertices that come after v in the chosen order.

During the search X is a list of vertices that form a clique; N is the set of common neighbors of X; and C is the set of common neighbors that come after the last vertex added to X.

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

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

Estimating the size of the search tree

If the number of choices only depends on the depth in the search $|C_i| = c_i$, the size of the tree is $|T| = 1 + c_0 + c_0c_1 + c_0c_1c_2 + \ldots + c_0c_1 \ldots c_{n-1}$. Usually this is not so. We label the vertices of the search tree by $[x_0, \ldots x_{l-1}]$ according to the choices made to reach them. The size of the tree can be estimated by picking at each step a choice uniformly at random, so that the probability of passing through vertex X is

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

where $f([x_0, \ldots x_{l-1}])=[x_0, \ldots, x_{l-2}]$ (parent of the node). We write $m(X) = 1$, if X is on the path, and $m(X) = 0$, if not. We estimate the size of the tree by computing

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

Claim: $E(N) = |T|$. Proof:

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

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

Example: 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	

In sudoku a partially filled $n \times n$ array is given. The array is divided into $p \times q$ subarrays. The task is to complete the array into a Latin square: each of the numbers $1 \ldots n$ must appear once in each row and column. Additionally each number must appear once in each subarray. (Usually $p = q = 3$ and $n = 9$.)

The most straightforward way of applying backtrack search would be to investigate the array square by square and always fill a square with a number that does not conflict with the numbers already in the array.

Example: Sudoku

A backtrack search can often be made more efficient by choosing the next value to be fixed to be one with the least number of alternatives. For example in sudoku we can choose to fill the square with the fewest admissible numbers.

If there are 0 choices, there is no solution; if there is 1 solution, we can deduce more about the solution without branching, which will limit alternatives later on.

By the way, sudoku could be expressed as a maximum clique problem: the vertex set would be formed by the n^3 row-column-value combinations, and two vertices would be connected by an edge if they are compatible (that is, they don't contain the same value in the same column for example). If there is a n^2 vertex clique in this graph, it corresponds to a solution.

Exact Cover

Given a set R and a set S of its subsets, can we express R as a disjoint union of sets in S?

$R = \{0, \ldots, n-1\}$. $S = \{S_0, \ldots, S_{m-1}\}$, where $S_i \subseteq R$ for all i. Does there exist $S' \subseteq S$, for which $\bigcup_{X \in S'} X = S$ and $S_i \cap S_j = \emptyset$, when $S_i,\ S_j \in S'$?

The cliques of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{0, \ldots, m-1\}$ and $\mathcal{E} = \left\{\{i, j\} : S_i \cap S_j = \emptyset\right\}$, correspond to partial solutions of the problem. We could use the AllCliques algorithm and test whether one of the maximal cliques is a solution.

Exact cover II

In the AllCliques algorithm the nodes are ordered. For subsets we take the decreasing lexicographical order. We denote by H_i the set of those subsets whose least element is i. The sets in H_i precede the elements of H_{i+1}.

In the AllCliques algorith C_l consists of vertices that come after all nodes already in the list and are neighbors of all vertices in the list – in this case, the subsets have no elements in common with any subset already on the list.

In the exact cover problem, since we must cover each element of the base set, we can decide to cover always the least element that has not been covered yet: if r is the least element not covered already, the choice set $C'_l = C_l \cap H_r$ suffices.

(Here too it could be more efficient to always consider the element next that has the least number of sets with which it could be covered.)

Sudoku as an exact cover problem

Sudoku can be mapped to exact cover in a straightforward manner. Each row-value, column-value, subarray-value and row-column combination must appear once. If the sets of rows, columns, subarrays and values are, respectively, $R = \{r_1, \ldots, r_n\}$, $C = \{c_1, \ldots, c_n\}$, $B = \{b_1, \ldots, b_n\}$, and $V = \{v_1, \ldots, v_n\}$, then the set to be covered is

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

If we write the value v_l into the square in row r_i, column c_j and subarray b_k, we cover the elements in

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

There are a total of n^3 such sets, and the cover will contain n^2 of them.

Bounding functions

In an optimisation problem the search tree may sometimes be pruned by estimating how good solutions can be found in a given branch.

Let profit (X) be the profit from solution X. Let $P(X)$ be the largest profit that can be obtained in the descendants of the partial solution X. Let $B(X)$ be an easily computable function for estimating $P(X)$ s.t. $B(X) \geq P(X)$.

If the best solution found so far is X', we are considering the partial solution X, and profit $(X') > B(X)$, we can prune this branch, since profit$(X') > B(X) \geq P(X)$, and among the descendants of X there can be no solution better than X'.

```
Bounding(X):
if X is a feasible solution:
        P ← profit(X)
            if P > OptP:
                OptP ← P
                OptX ← X
compute C_l
B ← B(X)
for each x ∈ C_l:
        if B ≤ OptP: # if we also prune on
            return # equality, the test must
                   # be here, since
                   # OptP may change
        Bounding(X + [x])
```

Rational knapsack

One method of forming bounding functions is relaxing some of the constraints of the original problem so that solving the optimization problem becomes easier.

Knapsack problem: Given n items with weights $w_1, \dots w_n$ and profits $p_1, \dots, p_n$. The capacity of the backpack is M. Maximize $P(x) = \sum p_i x_i$ subject to the constraints $x_i \in \{0, 1\}$ and $\sum w_i x_i \leq M$.

Rational knapsack problem: as above, but instead of requiring that $x_i \in \{0, 1\}$ require only that $0 \leq x_i \leq 1$.

Rational knapsack

Given n items with weights $w_1, \dots w_n$ and profits $p_1, \dots, p_n$. The capacity of the knapsack is M. Maximise $P(x) = \sum p_i x_i$ subject to $0 \leq x_i \leq 1$ and $\sum w_i x_i \leq M$.

For integer p_i, w_i, M the variables x_i will be rational. A greedy algorithm gives the optimum:

RKnap$(p_1, \dots, p_n, w_1, \dots, w_n, M)$
order the items s.t. $p_1/w_1 \geq p_2/w_2 \geq \dots \geq p_n/w_n$
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 $\sum p_i x_i$

Knapsack

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$(p_{l+1}, \dots, p_n, w_{l+1}, \dots w_n, M - CurW)$
for $c \in C_l$:
  if $B \leq OptP$
    return
  Knapsack3$(x + [c], CurW + w_{l+1} x_{l+1})$

This algorithm assumes that $\frac{p_1}{w_1} \geq \dots \geq \frac{p_n}{w_n}$.
In the book this pruning saved considerable effort for certain random problems (at best but not atypically (18953093→180).

Traveling salesman problem

Given $K_n = (\mathcal{V}, \mathcal{E})$, a complete (directed) graph on n vertices, and a cost function cost : $\mathcal{E} \mapsto \mathbb{Z}^+$. Find a Hamiltonian cycle X, for which cost$(X) = \sum_{e \in E(X)}$ cost(e) is minimum.
(A Hamiltonian cycle is a walk that visits every vertex once and returns to its starting point.)
A Hamiltonian cycle can be presented as a permutation of the vertices, and the cycle can be chosen to start at vertex 1. The tour 3 6 2 1 4 5 7 3 can thus be expressed as the list $[1, 4, 5, 7, 3, 6, 2]$.

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$(x) = 1$:
  $C_l \leftarrow \{2, \dots n\}$
else
  $C_l \leftarrow C_{l-1} \setminus \{x_{l-1}\}$
for each $c \in C_l$:
  TSP1$(x + [c])$

Bounding functions for the traveling salesman

The cost function can be represented as a matrix M, where m_{ij} is the cost of the (directed!) edge (i, j).

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

MinEdgeBound: Sum together the minimum value of each column (row); we must enter each vertex from some other node (go to some other node from each vertex).

ReduceBound: If k is subtracted from all elements in a row (column), the length of the tour goes down by k. Thus let c be the sum of the minimum elements in each column, and subtract from each element the minimum element in that column. From the resulting matrix compute r similarly by rows. After this each row and column contains at least one 0, and we obtain the lower bound $c + r$.

Bounding functions for the traveling salesman II

The lower bound corresponding to the partial solution $X = [x_0, \ldots, x_{l-1}]$ is obtained as follows: treat X as if it were one vertex, from which moving into vertex y costs cost $((x_{l-1}, y))$ and moving into which from vertex y costs cost $((y, x_0))$.

Remove from the matrix rows $x_0, \ldots, x_{l-2}$ and columns $x_1, \ldots, x_{l-1}$ and set $m_{l-1,0} = \infty$.

For example with the partial solution $[1, 2]$ we obtain

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

Thus we have reduced the problem to a directed traveling salesman problem with $n + l - 1$ vertices, to which the previously given bounds may be applied.

Bounds for the maximum clique problem

In the AllCliques procedure C_l is the set of common neighbors of the vertices in the partial solution that come after the vertices in the partial solutions in the chosen order.

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

Bounds from the graph coloring problem: If the vertices can be colored with k colors so that no edge has two endpoints of the color, the largest clique can have at most k vertices (all vertices in a clique are neighbors and thus of different color).

Bound: color the graph induced by the vertices in C_l. If this can be done with k colors, $B(X) = |X| + k$.

The graph coloring problem is computationally difficult. A bound can be obtained by a greedy algorithm: label each vertex in turn with a positive integer as small as possible. Or we can start by coloring the vertices and during the search count the number of distinct colors in C_l.

Branch and bound

Up to now $x \in C_l$ have been visited in an arbitrary order. It could be better to compute for each x the bound $B(X + [x])$ and examine the most promising alternatives first. In this manner we may find good solutions for pruning the search.

```
BranchAndBound(X):
if X a feasible solution:
    P ← profit(X)
        if P > OptP:
            OptP ← P
            OptX ← X

compute C_l

v ← [ ]
for each x ∈ C_l:
    compute B_x ← B(X + [x])
    v ← v + [(x, B_x)]
order v in decreasing order of B_x

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

Dynamic programming in the maximum clique problem

Given $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, n\}$. Precompute $N_v = \{u \in \mathcal{V} : \{u, v\} \in E\}$ and $G_v = \{u \in \mathcal{V} : u \geq v\}$. We denote by c_v the size of the maximum clique in G_v. Now $c_{i-1} \in \{c_i, c_i + 1\}$ and $c_{i-1} = c_i + 1$ if and only if G_{i-1} contains a clique of $c_i + 1$ vertices (which must include vertex $i - 1$).

for $i = n$ downto 1:
    $found \leftarrow$ false
    MaxClique$([i], G_i \cap N_i)$
    $c_i \leftarrow OptP$
MaxClique(X, N):
if $|X| > OptP$:
    $OptP \leftarrow |X|$
    $OptX \leftarrow X$
    $found \leftarrow$true
    return
if $|X| + |N| \leq OptP$:
    return
for $x \in N$:
    if $|X| + c_x \leq OptP$:
        return
    MaxClique$(X + [x], N \cap N_x)$
    if $found$:
        return

Heuristic methods

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

- when the search space is too large for backtrack search
- obtain good solutions by trial and error and repeatedly making small changes to earlier solutions
- suitable for optimisation problems (if a good solution suffices) and search problems, but usually not for generation or enumeration problems

Optimisation problem

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

$P(x)$ is the objective function, and $g_j(x) \leq 0$ are constraints. Any $x \in X$ is a solution. If additionally $g_j(x) \leq 0$, then x is feasible. If $P(x) \geq P(x')$ for all $x' \in X$, $g_j(x') \leq 0$, the solution x is optimal.

The penalty function method makes infeasible solutions feasible, which may make designing the search easier. It can also be used to convert a search problem into an optimisation problem.

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

where $\Phi(y) = 0$, when $y \leq 0$, and $\Phi(y) > 0$, when $y > 0$. We may choose a sufficiently large value for μ at the start or increase μ little by little.

Heuristic methods in general

The neighborhood of a solution is a central concept in many heuristic methods. The neighborhood of a solution x is $N(x) \subseteq X$. The neighborhood heuristic $h_N(x)$ returns a feasible solution in the neighborhood of k or $Fail$.

GenericHeuristicSearch:
$x \leftarrow$ some feasible $x \in X$
$BestX \leftarrow x$
while not *termination condition*:
    $y \leftarrow h_N(x)$
    if $y \neq Fail$
        $x \leftarrow y$
        if $P(x) > P(BestX)$
            $BestX \leftarrow X$
return $BestX$

Some simple heuristics

1. find the feasible $y \in N(x) \setminus \{x\}$, that maximises $P(y)$; return y or $Fail$, if there is no feasible neighbor
2. find the feasible $y \in N(x)$, that maximises $P(y)$; if $P(y) > P(x)$, return y, else $Fail$
3. find some feasible $y \in N(x)$
4. find some feasible $y \in N(x)$; if $P(y) > P(x)$, return y, else $Fail$

Equitable graph partition

Given a $2n$-vertex complete graph $G = (\mathcal{V}, \mathcal{E})$ and a cost function $\text{cost} : \mathcal{E} \mapsto \mathbb{Z}^+ \cup \{0\}$.

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

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

Example algorithm: Let the solution space X be the set of those partitions $[\mathcal{V}_0, \mathcal{V}_1]$ of $\mathcal{V}$ for which $|\mathcal{V}_0| = |\mathcal{V}_1| = n$. The neighborhood $N(x)$ of a partition x is the set of those partitions that can be obtained from x by moving one element from each set to the other. The heuristic $h_N(x)$ could be *steepest ascent*: find the best neighbor y; if the value of the objective function improves, return y, else $Fail$.

Neighborhood heuristics

When maximising, in each iteration:

Steepest ascent: choose the feasible neighbor of x that maximises the objective function, until there is no neighbor with a better objective function value

Hill-climbing: choose some feasible neighbor of x that improves the value of the objective function until there is none

Both of these stop in the first *local optimum*. A local optimum is a solution x such that $P(x) > P(y)$ for all feasible $y \in N(x)$.

To a certain extent local optima can be eliminated by considering a larger neighborhood, but that alone rarely solves the problem; also a larger neighborhood can cause other problems, like making computation of $h_N(x)$ more expensive.

Neighborhood heuristics II

Great deluge: In each iteration choose a feasible neighbor $y \in N(x)$, for which $P(y) \geq W$, where W is the water level. Increase W every now and then until there are no feasible neighbors left.

Record-to-record travel: In each iteration choose a feasible neighbor $y \in N(x)$, for which $P(y) \geq P(BestX) - D$, where D is a constant.

Simulated annealing

Simulated annealing is based on an physical model of cooling metal.

$h_N(x)$: Choose a random $y \in N(x)$. Let $\Delta P = P(y) - P(x)$. If $\Delta P \geq 0$, return y. If $\Delta P < 0$, return y with probability $e^{\Delta P/T}$, where T is the current temperature of the system; else $Fail$.

At first T is relatively large, so moves that worsen the solution are accepted relatively often. As the search proceeds the temperature is decreased so that worsening moves are accepted more and more rarely.

A simple cooling schedule is obtained by setting in each iteration $T \leftarrow \alpha T$, where α is slightly less than 1.

Tabu search

A heuristic, where the feasible neighbor of x with the largest objective function value is chosen, may move out of the local optimum but often moves back in the next iteration. Therefore tabu search:

$h_N(x)$: Choose the feasible neighbor of x with the largest objective function value; however, do not undo any change caused by a move in the last l iterations.

change (x, y) describes the change made when moving from x to its neighbor y.

TabuSearch

$TabuList \leftarrow [\,]$
choose a feasible $x \in X$
while not *termination condition*:
  $T = \{y : y \in N(x), \text{change}(x, y) \in TabuList\}$
  $N \leftarrow N(x) \setminus T$
  find the feasible $y \in N$ with the largest $P(y)$
  $x \leftarrow y$
  append change (y, x) to $TabuList$
  remove from $TabuList$ all except the l most recent elements
  if $P(X) > BestP$
    $BestP \leftarrow P(X)$
    $BestX \leftarrow X$

Choosing the neighborhood

The combination of the neighborhood and solution space can be interpreted as a directed graph, whose vertices are the solutions. There is an edge from vertex x to vertex y, if $y \in N(x)$.

Properties of a good neighborhood:

1. Every solution – or at least the optimum solution – is reachable from every other solution.
2. The neighborhood is relatively small, and the objective function values of a solution and its neighbors exhibit at least some correlation.
3. The neighborhood is sufficiently large that every solution – or at least the optimum solution – is reachable from every other solution with a relatively small number of moves.

Smoothness of the objective function

It is advantageous if the objective function values are high for solutions that are near the maximum. Let $x_i \in \{0,1\}$ and $N(x) = \{y \in \{0,1\}^n : \text{dist}(x,y) = 1\}$.

Compare.
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. $\max 2n \prod_i x_i - \sum_i x_i$

Large "valleys" or "plateaus" in the objective function can cause problems. The chosen solution space and neighborhood are crucial.

Genetic algorithms

Instead of maintaining one current solution, we may maintain a whole population. New solutions are obtained from the old ones by crossover and mutation.

In crossover two solutions are combined to obtain two new ones. In mutation the solution x is replaced by one of its neighbors; $x \leftarrow h_N(x)$.

GeneticAlgorithm

GeneticAlgorithm:
choose the original population P
while not *termination condition*:
    $Q \leftarrow P$
    compute the list of pairs for crossover R
    for $(w,x) \in R$
        $(y,z) = \text{crossover}(w,x)$
        $y \leftarrow h_N(y)$
        $z \leftarrow h_N(z)$
        $Q \leftarrow Q \cup \{y,z\}$
    $P \leftarrow$ popsize best individuals from Q
    $b \leftarrow$ best individual in Q
    if $P(b) > P(BestX)$
        $BestX \leftarrow b$

Crossover and natural selection

Crossover for lists $A = [a_1, \ldots, a_n]$ and $B = [b_1, \ldots, b_n]$ can be carried out for example as follows:

single-point crossover Choose $1 \le j < n$. The descendants are
$C = \left[a_1, \ldots a_j, b_{j+1}, \ldots b_n\right]$ and
$D = \left[b_1, \ldots b_j, a_{j+1}, \ldots a_n\right]$.

two-point crossover Choose $1 \le j < k \le n$. The descendants are
$C = \left[a_1, \ldots, a_j, b_{j+1}, \ldots, b_k, a_{k+1}, \ldots, a_n\right]$ and
$D = \left[b_1, \ldots, b_j, a_{j+1}, \ldots, a_k, b_{k+1}, \ldots, b_n\right]$.

uniform crossover Choose $S \subseteq \{1, \ldots, n\}$. In the descendants $c_i = a_i$ and $d_i = b_i$, if $i \in S$; otherwise $c_i = b_i$ and $d_i = a_i$.

To combine permutations α and β we may choose $1 \le j < k \le n$, and

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

The results of crossover may not always be feasible solutions. We may
1) use the penalty function method
2) design a special crossover function, with which the descendants are always feasible.

Pairing: the solutions in the population may be paired using various criteria, such as by first ordering them by objective function value. One may also generate more offspring from fitter individuals.

Steiner triple systems

A Steiner triple system is a set system $(\mathcal{V}, \mathcal{B})$, where $\mathcal{B}$ consists of 3-subsets of $\mathcal{V}$, and each 2-subset of $\mathcal{V}$ occurs as a subset in exactly one $b \in B$. An STS is a partition of the complete graph into triangles. E.g.

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

Hill-climbing and Steiner triple systems

The point $v \in V$ is *live*, if it is incident to edges that are not contained in any triangle $b \in B$. The edge $\{u, v\}$ is live, if it appears in no $b \in B$.

Stinson'sAlgorithm(v):
$\mathcal{V} \leftarrow \{1, \ldots, n\}$
$\mathcal{B} \leftarrow \emptyset$
while $|\mathcal{B}| < v\,(v-1)\,/6$:
    choose a live point x
    choose live edges $\{x, y\}$ and $\{x, z\}$
    if $\{y, z\}$ is live:
        $\mathcal{B} \leftarrow B \cup \{\{x, y, z\}\}$
    else
        find the block $\{w, y, z\}$ in $\mathcal{B}$
        $\mathcal{B} \leftarrow B \cup \{\{x, y, z\}\} \setminus \{\{w, y, z\}\}$

Knapsack and simulated annealing

Knapsack problem: Given n items with weights $w_1, \ldots w_n$ and profits $p_1, \ldots, p_n$. The capacity of the knapsack is M. Maximise $P(x) = \sum p_i x_i$ subject to $x_i \in \{0, 1\}$ and $\sum w_i x_i \le M$.

Choose the neighborhood:
$N(x) = \{y \in \{0,1\}^n : \text{dist}\,(x, y) = 1\}$.
A random neighbor y can be obtained by flipping a random x_j.

If $x_j = 0$, the change of the objective function is $\Delta P = +p_j$, and the new neighbor is accepted, if it is feasible. If $x_j = 1$, then $\Delta P = -p_j$, and the new neighbor is accepted with probability $e^{-p_j/T}$.

At first set T so that a considerable part of the worsening moves are also accepted; e.g. $4 \max_i p_i$, and after each iteration set $T \leftarrow \alpha T$. In the book best results were obtained with $\alpha = 0.9999$.

Knapsack and tabu search

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

We won't maximise the objective function(?!!), but rather

1. add to the knapsack item i, that has the largest p_i/w_i ratio of the items that are not tabu, not in the backpack and fit into the backpack

2. if no such item exists, remove from the knapsack item i that has the smallest p_i/w_i ratio of the items that are not tabu but are in the backpack.

3. Add i to the tabu list.

Graph coloring

What is the least number of colors sufficient for coloring the vertices of graph $G = (\mathcal{V}, \mathcal{E})$ so that no edge has two endpoints of the same color?

Partition the vertices into color classes $\mathcal{V}_1 \ldots \mathcal{V}_k$ e.g. by a greedy algorithm. After having obtained a k-coloring, search for a $k-1$-coloring as follows.

Take as the objective function $\max \sum_i |\mathcal{V}_i|^2$. This guides the search so that some colors are used to color many vertices and others only a few. If the number of vertices in some part goes to zero, a $k-1$-coloring has been found, and we can start looking for a $k-2$-coloring etc.

Schur numbers

The Schur number $s(m)$ is the largest integer s, such that the integers $X = \{1, \ldots, s\}$ can be partitioned into m parts so that no part contains two (not necessarily distinct) elements and their sum.

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

In searching for a sum-free partition of $1 \ldots s$, the most obvious choice for an objective function would be the number of sums appearing in the parts, but it is better to maximise
$\max c_1 f_1 + c_2 f_2$, where $c_1 \gg c_2$ and
$f_1 = \max t$, such that no part contains the sum t
$f_2 = \sum_{i=1}^{m} \sum_{t,u \in X_i} g(t,u)$

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

Here the purpose of f_2 is only to guide the search to a promising direction.

Traveling salesman problem

Solve the traveling salesman with genetic algorithms.

The n-opt neighborhood: remove n edges from the cycle and add n edges to again obtain a cycle.

The crossover function could be defined as follows: cross the two permutations by some crossover method for permutations, and then apply the steepest descent method with, say, the 2-opt neighborhood.

We could also simply crossover the permutations and embed steepest descent into the objective function. Then the fitness of a permutation would be evaluated by first applying steepest descent to it and only then computing the length of the cycle.

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 (bijections $\pi : \mathcal{X} \mapsto \mathcal{X}$) 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 ← |G| / |H|
T ← ∅
for g ∈ G:
    for t ∈ T:
        if t^{-1} g ∈ H:
            goto skip
    T ← T ∪ {g}
    if |T| ≥ r:
        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 rank$(g(A))$ < 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 and 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.

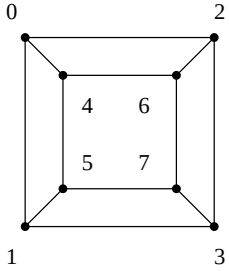
Computer representations of a permutation group

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 $\alpha\alpha\alpha\alpha\beta\beta\beta = \alpha\beta\alpha\alpha$.

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

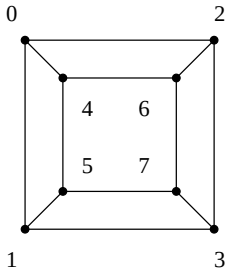
Let us apply the idea recursively:

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

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$(n, g, G = [\mathcal{U}_0, \ldots, \mathcal{U}_{n-1}])$:
for $i \leftarrow 0$ to $n-1$:
    if there is a $h \in \mathcal{U}_i$, for which $h(i) = g(i)$:
        $g \leftarrow h^{-1}g$
    else:
        return i
return n

Computing a Schreier-Sims representation

enter$(n, g, G = [\mathcal{U}_0, \mathcal{U}_1, \ldots, \mathcal{U}_{n-1}])$:
$i \leftarrow$test$(n, g, G = [\mathcal{U}_0, \mathcal{U}_1, \ldots, \mathcal{U}_{n-1}])$
if $i = n$
    return
$\mathcal{U}_i \leftarrow \mathcal{U}_i \cup \{g\}$
for $j = 0$ to i:
    for $h \in \mathcal{U}_j$:
        enter(n, gh, G)

main:
for $i \leftarrow 0$ to $n-1$:
    $\mathcal{U}_i \leftarrow \{\mathbf{I}\}$
for $\alpha \in \Gamma$:
    enter$(n, \alpha, G = [\mathcal{U}_0, \ldots, \mathcal{U}_{n-1}])$
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 shortest 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 $\pi \in \text{Sym}(\mathcal{V})$ we obtain the incidence matrix

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

$\text{Num}_\pi(\mathcal{G})$ 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:

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

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

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

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 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 $G = (\mathcal{V}, \mathcal{E})$, 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

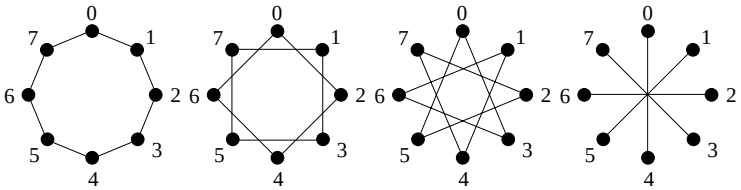
The Ramsey number $R(k, l)$ is the least integer n, for which all n-vertex graphs contain a k-vertex clique or an l-vertex independent set. We show that $R(3, 4) > 8$ by finding an 8-vertex graph with no 3-vertex clique and no 4-vertex independent set.

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 $\mathcal{E} = O_3 \cup O_4$ and $\mathcal{E} = O_1 \cup O_4$ 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 $\mathcal{E} = \{\{v, v+d \pmod{120}\} : d \in S, v \in \mathcal{V}\}$, $\mathcal{V} = \{0,\ldots,119\}$ and $S = \{2,3,6,7,13,15,17,18,19,20,22,23,28,29,31,33,41,42,43,45,48,52,53,54,60\}$, 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

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

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 $|G_x| = |G_y|$, 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:

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

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
$\frac{1}{2}(243 + 27) = 135$.