Combinatorial Algorithm Note

author: Ray Zhan
email : ray1710852730@163.com

Before start

Based on Prof. Carlos's Combinatorial Algorithm Course. no proof version.

'*' means it is not covered in detail in the course.

Disclaimer : The link in this article is directed to other websites, which are **NOT** guaranteed to be 'safe', and the **link maybe maliciously modified by malicious 're-uploader'**. Please check SHA-256 () written here. And please read/download this article from trusted source

(author's github account : https://raylern.github.io/Github_Webpage <--Do **NOT** click this directly, please copy-paste the text url to your browser!!!).

Combinatorial Structures & Concepts

Graphs

Graphs		
undirected graph	$G = (V, E)$	
directed graph	$G = (V, E), E = \text{set of tuples}$	
colored graph	$G = (V, E, f), f : V \mapsto Colors$	
hypergraph/set system	$G = (V, B)$	

- complete graph
- tree
- clique
- hamiltonian cycle
- induced subgraph : $G[W]$ is the graph induced by a set of vertices W
- steiner triple system, STS

  $STS(3) = \{\{0, 1, 2\}\}$

  $\exists STS(n) \iff n\ mod\ 6 = 1\ or\ 3$
- *Latin square

  $L = n \times n \times \Sigma = \{(row, col, content)\}$

  $L = \text{set system}(V, B), V = X \times \{1, 2, 3\}, B = \{\text{3-subsets of } V\}$

  Example,

$$\begin{pmatrix} a & b \\ b & a \end{pmatrix}$$

- *transversal design

  $TD = \text{set system}(V, B, X), |V| = 3n, B = \{\text{3-subsets of } V\}, X = \{X_1, X_2, X_3\} \text{ a partition of V}$

  Example,

  $(V = \{1, 2, 3, 4, 5, 6\}, B = \{\{1, 3, 5\}, \{2, 3, 6\}, \{1, 4, 6\}, \{2, 4, 5\}\}, X = \{\{1, 2\}, \{3, 4\}, \{5, 6\}\})$

data structures:

- incidence matrix : a $|B| \times |V|$ matrix

  Example, $V = \{1, 2, 3, 4\}, B = \{\{1, 2\}, \{2, 3\}, \{2, 3, 4\}\}$

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

- adjacency matrix
- adjacency list

- point-block incidence graph

Graph Isomorphism

- isomorphism / isomorphic
- automorphism

		Examples
invariant	an encoding that $G_1 \cong G_2 \implies I(G_1) = I(G_2)$	sorted degree sequence, number of cliques
certificate	an encoding that $G_1 \cong G_2 \iff Cert(G_1) = Cert(G_2)$	tree certificate
canonical form	representative of isomorphism class	matrix canonical form

- invariant inducing function

  $D: V \to \{0, 1, \dots, k\}$ is an invariant inducing function, if $\phi_D(G) = [|B[0]|, |B[1]|, \dots, |B[k]|]$ is an invariant (under isomorphism), where $B[i] = \{v \in V : D(v) = i\}$.

Vertex Partition

- induced vertex partition

  Given function $D : F \times V \to \{0, 1, 2, \dots, k\}$, F is a family of graphs on vertices V .

  partition induced by D is $B = [B[0], \dots, B[k]], where\ B[i] = \{v : D(G, v) = i\}$
- discrete partition : finest
- unit partition : only 1 block
- equitable partition :

  graph $G = (V, E)$, adjacency / neighbor of u, $N_G(u) = \{x \in V : \{u, x\} \in E\}$.

  partition P is equitable $\iff$ Given any i, $\forall u, v \in B[i], \forall j, |N_G(v) \cap B[j]| = |N_G(u) \cap B[j]|$.
- matrix associated to an equitable partition :

  Given equitable partition P with k blocks, matrix $M_P \in \mathbb{N}^{k \times k}$, $M_P[i, j] = |N_G(v) \cap P[j]|$, where $v \in P[i]$.

  > Well-defined by definition of equitable partition
- (ordered) partition refinement

Sets

- set
- k-subset
- directed graph $G = (V, E)$
- hypergraphs/set system $S = (V, B)$

Bit vectors

- gray code
- binary reflected gray code
- hamming distance
- (totally) balanced sequence

*Code

- *(block) code :

  q-ary (block) code C of length n is a non-empty subset of $(\mathbf{\Sigma}_q)^n$, where $\mathbf{\Sigma}_q$ is a finite alphabet of size q

  Example, $C = \{000, 001, 201\} \subseteq \mathbb{Z}_3^4$ is a ternary code of length 3

  Equivalence of code :

  $C_1 \cong C_2$

  $\iff$ C_1 can be obtained by permuting coordinates and permuting $\mathbf{\Sigma}_q$ from C_2

  Example:

  $$C_1 = \{000, 011, 201\} \xrightarrow{position 1 \leftrightarrow 2} \{000, 101, 021\} \xrightarrow{value 1 \leftrightarrow 2} \{000, 202, 012\} = C_2$$

- *linear code :

  (linear) code C of length n over field F is a subspace of F^n

  Representation of linear code :

  1. generator sets
  2. generator matrix

  Equivalence of linear code :

  1. Permutation Equivalent

     $C_1 \cong_\pi C_2$

     $$\iff \exists \text{ permutation matrix } P, s.t.$$
     $$\forall G_1, \text{ generator matrix of } C_1, G_1 P \text{ is a generator matrix of } C_2.$$

     $\iff$ rearranging columns of generator matrix of G_1 gives generator matrix of G_2

  2. Monomially Equivalent

     <u>**Def** (Monomial matrix)</u> square matrix A is monomial $\iff$ A has exactly 1 nonzero entry in each row and column.

     $C_1 \cong_M C_2$

$\iff \exists \text{ monomial matrix } M, s.t.$

$\forall G_1, \text{ generator matrix of } C_1, G_1 M \text{ is a generator matrix of } C_2.$

$\iff$ scalar (in field F) multiple and permutation of columns of generator matrix of G_1 gives generator matrix of G_2

Example :

> Remark :
>
> $\cong_\pi \implies \cong_M$

3. Code Equivalent

   $C_1 \cong C_2$

   $\iff \exists \text{ monomial matrix } M, \exists \text{ automorphism of } F_q, \sigma,$ s.t.

   $\forall G_1, \text{ generator matrix of } C_1, G_1 M\sigma \text{ is a generator matrix of } C_2.$

   $\iff$ scalar (in field F) multiple, permutation of columns of generator matrix of $G_1$and automorphism of alphabet, gives generator matrix of G_2.

   Example :

   $< 001, 211, 120 > \cong < 002, 122, 210 >$

Types of Problems

- **decision problem**
- **search problem**
- **optimization problem**

  (generic optimization)

```
instance: A finite set S
          an objective function P : S → Z.
          a finite set of feasibility functions Fi : S → Z, 1 ≤ i ≤ m.
solution: x in S, s.t. P(x) maximal, Fi(x)≥0
```

Complexity Class

P	decision problem	solve in polynomial time	tree encoding
NP	decision problem	**verify** in polynomial time	
NP-complete	NP (decision problem)	any other NP problem can be reduced to it by polynomial transform	Knapsack(Decision), Max clique(Decision)

NP-hard	any problem	problem that can be **Turing-reduce** to NP-complete problem	Knapsack(Optimization)

- *Polynomial transformation : $P_1 \propto P_2 \iff$ any instance of P_1 can be transformed to instance of P_2 in polynomial time and they share the same "yes"/"no" answer.
- *Turing reduction : P_2 with algorithm A_2, $P_1 \propto_T P_2 \iff \exists$ algorithm A_1 that solves P_1 and uses A_2 as sub-routine.

Searching techniques

- backtrack
    - state-space tree / search tree
    - choice set
    - pruning
    - bounding function
    - branch & bound : exploring the nodes in Choice Set with priority given by bounding function (explore nodes with better potential first)
- exhaustive search
- heuristic search
    - neighbourhood
    - neighbourhood search
    - hill climbing
    - simulated annealing
    - tabu search
    - genetic search

> Remark:
>
> | | |
> |---|---|
> | backtrack | any problem |
> | backtrack with bounding | optimization problem |
> | branch & bound | optimization problem |
> | heuristic | solution "close to" optimal, optimization problem |

Combinatorial Problems & Algorithms

Generation

All Subsets

```
AllSubsets: find all subsets of a given set
```

characteristic vector : !! Notice n=3, {1,2}->[1,1,0]

lexicographic ordering

- ranking: easy
- unranking: easy
- successor:

  ```
  for i= n -> 0
      if i not in S
          S.add(i)
          remove all number in S bigger than i
  ```

minimal change ordering

- ranking:

  ```
  1.concatenate top 0
  2.add 2 consecutive bit
  3.get correponding bit in rank in binary
  ```

  Example: {1,2,5} -> 11001 -> 011001 -> 10101 ->21

- unranking:

  ```
  1.make rank to binary
  2.add all bits left to current bit (including current bit)
  3.get correponding bit in characteristic vector
  ```

  Example: 21 -> 10101 -> 11001 -> {1,2,5}

- successor:

  ```
  1a. if even weight -> flip last bit;
  1b. if odd weight -> right to left, flip bit after the first 1
  ```

  Example: {1,2,5} -(odd)-> {1,2,4,5}

k-subset

```
AllSubsets: find all subsets of a given set
```

lexicographic ordering

- ranking: $rank(S) = \sum_{i=1}^{k} \sum_{j=s_{i-1}+1}^{s_i - 1} \binom{n-j}{k-i}$
- unranking: left to right, judge whether (x=1 to n) is at that place
- successor: right to left, increase first number that can be increased, reset number behind it to its successors.

  ```
  for i= k -> 0
      if S[i] < n-i
          S[i]+=1
          reset number after position i
  ```

Permutation

lexicographic ordering

- successor:

  ```
  1. from right to left, find longest decreasing suffix
  2. exchange non-decreasing element with its predecessor in the suffix
  3. sort suffix in increasing order
  ```

  Example: [1,3,<u>5,4,2</u>] -> [1,4,<u>5,3,2</u>] -> [1,4,2,3,5]

- *rank:

  $$rank(\pi, n) = (\pi[0]-1)(n-1)! + rank(\pi', n-1)$$
  $$where\ \pi'[i] = \begin{cases} \pi[i+1]-1 & \text{if } \pi[i+1] > \pi[0] \\ \pi[i+1] & \text{if } \pi[i+1] < \pi[0] \end{cases}$$

  Example:

  [1,3,5,4,2] -> 0+[2,4,3,1] -> 0+6+[3,2,1] -> 0+6+4+[2,1] -> 0+6+4+1+0=11

- *unrank:

  factorial representation of a number, $r = \sum_{i=1}^{n-1}(d_i \times i!), 0 \le d_i \le i$

  ```
  p[n-1]=1
  for i=n-2 to 0
      calculate d_i
      p[i]=d_i+1
      for j=i+1 to n-1
          if p[j]>d_i, then p[j]++
  ```

  Example:

  unrank(11,5)--> [?,?,?,?,1], r'=0+6+4+1 --> [?,?,?,2,1], r''=0+6+4 --> [?,?,3,2,1], r'''=0+6 --> [?,2,3(+1),2(+1),1], r=0 --> [1,2(+1),4(+1),3(+1),1(+1)]=[1,3,5,4,2]

Backtrack

Knapsack

```
Instance: K=(P,W,M), P=list of profits, W=list of weight, M=maximum weight
Solution: a list of items taken S, total weight <= M
(Decision) Is there a solution S with given profit p?
(Optimization) Find solution S with maximum profit
```

Backtrack

iterate over all combinations of choices of items

Backtrack with Choice set

$C_l = \{\text{items with larger No. and weight not exceeded}\}$, current weight maintained

Backtrack with bounding

sort items according to $profit/weight$

using $RKnapsack$ as an upper bound

All Cliques

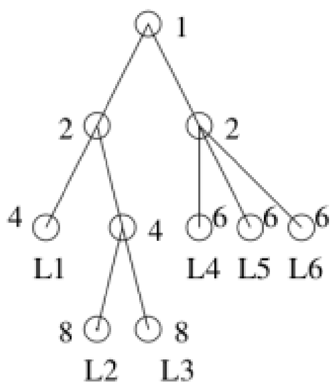
```
Instance: a graph G
Solution: a list of all cliques in Graph
```

- $C_l = A_{X[l-1]} \cap B_{X[l-1]} \cap C_{l-1}$, $A_{X[l-1]}$={vertices adjacent to $X[l-1]$}, $B_{X[l-1]}$={vertices > $X[l-1]$} for eliminating repeated cliques
- $N_l = A_{X[l-1]} \cap N_{l-1} = \{\text{nodes that can potentially added to form bigger clique}\}$ partially 'maximal' clique $\iff N_l = \emptyset$

Size of Backtrack Tree

```
Instance: a backtrack algorithm/tree
Solution: number of nodes estimated
```

- one path: go through a certain path estimate corresponding to branch number



$estimate(L2) = 1 + 2 + 4 + 8 = 15$

- several paths: calculate the average, or calculate the weighted average by the ratio of probability of selecting that path.

Theorem the expectance of estimated tree size = real size

Exact Cover

```
Instance: (X,B), B={subsets of X}
Solution: a exact cover S⊆B, S consists of disjoint sets, ∪S=V
```

Transforms to clique problem

$V = B, (B_i, B_j) \in E \iff B_i \cap B_j = \emptyset (disjoint)$

$\text{exact cover } S \implies \text{ a clique in } G = (V, E) \text{ that covers all } V$

Backtrack

The idea is the same as solving clique problem.

But now, we do NOT need to find all the cliques, what we need is a clique covering all numbers if exists.

- First, sort the blocks $V = B$ in lexicographic **decreasing** order
- $C_l' = A_{X[l-1]} \cap G_{X[l-1]} \cap C_{l-1}'$, $A_{X[l-1]}$={vertices adjacent to $X[l-1]$}, $G_{X[l-1]}$={vertices > $X[l-1]$} (since vertices are subsets, $>=>_{lex}$) for eliminating repeated cliques, as usual.
- maintaining variable $r = \text{least number that is NOT covered in current cover}$
- $C_l = C_l' \cap H_r$, $H_r = \{\text{blocks whose least element is r}\}$

Maximum Clique (with bounding)

```
Instance: a graph G
Solution: maximum size of clique exists in Graph
```

The same as all clique algorithm but with bounding.

```
...
Compute choice set C_l
b = B(X)
for each x in C_l
    if b<= currOpt then return
    x_l = x
    next iteration
```

general bound

$B(X) = |X| + bounding(G, X)$, $bounding(G, X)$ is yet to be designed

> Remark: "$b \le currOpt$" need to be checked every cycle, since during next iteration, $currOpt$ may decrease and make "$b \le currOpt$" satisfied in later cycles

size bound

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

greedy color

Lemma G has a k-coloring $\implies$ G don't have clique of size >k

a greedy heuristic algorithm for finding a small coloring (NOT solving coloring problem, but finding a upper bound for color needed)

```
GreedyColor(V,E)
    k=0 //colors used currently
    for i=0 to |V|-1
        find an existing color that all neighbours of i is NOT in that color
        if not found
            add 1 color
            color i with new color
            k++
    return k
```

sampling bound

run $GreedyColor$ before hand

$B(X)=|X|+|\{Color[x]:x\in C_l\}|$

greedy bound

$B(X)=|X|+GreedyColor(G[C_l])$, $G[C_l]$ is the induced graph

Heuristic Search

Generic Optimization

```
instance: A finite set S
          an objective function P : S → Z.
          a finite set of feasibility functions Fi : S → Z, 1 ≤ i ≤ m.
solution: x in S, s.t. P(x) maximal, Fi(x)≥0
```

neighbourhood function

computing neighborhood of a (partial) solution

> remark:
>
> 1. N(X)=(partial) solutions "close to" current solution X
> 2. fast to compute

> 3. should be designed that possible to achieve real optimal with finite steps of applying neighborhood function
>
> N(X) may give infeasible solutions

neighbourhood seach

4 classic strategies

- exhaustive
  - steepest ascend
  - best neighbour
- randomized
  - random improvement

    ```
    return the random result if it improves current solution
    else "fail"
    ```

  - random feasible

> remark:
>
> 1. should be designed to only give feasible solution or "fail"

Generic Heuristic

```
GenericHeuristic(c_max)
    select X = initial solution
    while c < c_max
        Y = NeighbourhoodSearch(X)
        if Y is not fail
            X=Y
            if Y improves OptX
                OptX=Y
        c++
    return OptX
```

Uniform Graph Partition

```
Instance: a graph G=(V,E) on 2n vertices
          cost function, c: E->N
Solution: partition (X,Y) of V, |X|=|Y|=n, Cost(X,Y)=sum of c(x,y) is minimal
```

$$neighborhood((X,Y)) :=$$

$$\{(A,B) : (A,B) \text{ is obtained by exchanging 1 element in X with 1 element in Y }\}$$

> random initial partition can be achieved by
>
> 1. random n-subset
> 2. shuffle

Hill Climbing

continuously applying **steepest ascend** neighborhood search, exit if "fail"

Simulated Annealing

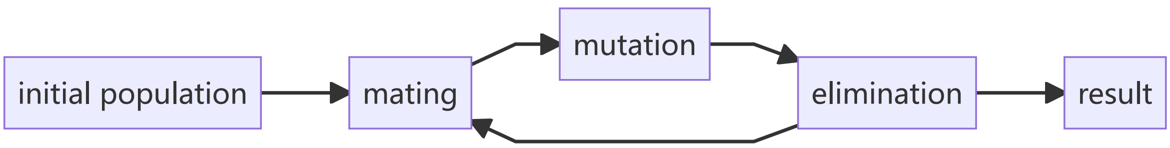
using **randomized** neighborhood search, allowing downward move by probability according to current temperature

Tabu Search

register change in **Tabu list** to forbid cycling for short term

> Remark:
>
> to be designed: change(X,Y)
>
> additional data structure: Tabu list (queue)

Genetic algorithm



mutation scheme

applying neighborhood search on each member individually

elimination scheme

- balanced scheme (keep best $popSize$-people in old and new generation)

mating scheme

- grouping scheme
  - randomly evenly grouping (randomly partition population into pairs)
- reproducing scheme
  - "better" parents can have more kids
  - evenly reproducing

cross over

for linear object

variable j=random cross over point

Example:

j=3, x=1101001, y=0101111

crossover(x,y)=110 0111, 0101001

partially matched crossover

for permutation

variable j,k=random match segment

Example:

j=3, k=6, x=[1,2,3,4,5,6,7,8], y=[3,7,1,5,8,6,2,4]

pmc(x,y)=[1,2,3,5,8,6,7,4], [3,7,1,8,4,6,2,5]

for both x and y : 1. exchange $5 \leftrightarrow 4$, 2. exchange $4 \leftrightarrow 8$, 3. exchange $6 \leftrightarrow 6$

Travelling Salesman Problem

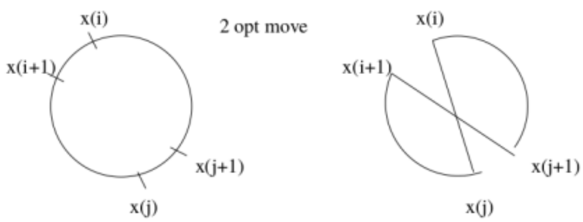
```
instance: complete graph Kn
          cost function c: VxV->R
solution: hamiltonian circuit X, C(X) is minimized
```

2-opt move

MGK recombine

- Mutation/Neighborhood search:

  2-opt move

  Gain in applying a 2-opt move:

$$
\begin{aligned}
G(X,i,j) &= C(X) - C(X_{ij}) \\
&= c(x_i, x_{i+1}) + c(x_j, x_{j+1}) - c(x_{i+1}, x_{j+1}) - c(x_i, x_j)
\end{aligned}
$$

  steepest ascend among all 2-opt moves

- Initial population is done by Permutation Unrank
- Mating:
  1. Partially match over
  2. **MGK recombine**

     variable j=random position, l=random length the segment S is first copied to the beginning of a new child, then completed to a feasible solution by appending the nodes of the other parent not in S in the order in which they appear.

Example:

j=2, l=4, x=[1,2,3,4,5,6,7], y=[2,1,5,3,7,6,4]

mgk(x,y)=[1,5,3,7,2,4,6], [2,3,4,5,1,7,6]

Knapsack

neighborhood is is defined by Hamming distance = 1 (drop or pick an item)

simulated annealing

randomized neighborhood search
drop item = downward move
add item = upward move

tabu search

"best neighbor" neighborhood search, "best" according to maximal $(-1)^{x_i}\frac{p_j}{w_j}$

Steiner Triple System

Stinson's Algorithm (hill climbing)

```
instance: n
solution: construct an STS(n)
```

Lemma $STS(n)=(V,B)$, then every points in V occurs in exactly $r=\frac{n-1}{2}$ blocks and $|B|=\frac{n(n-1)}{6}$

Theorem $\exists STS(n) \iff n\ mod\ 6 = 1\ or\ 3$

```
Switch()
    randomly choose live point x
    randomly choose y,z, s.t. (x,y), (x,z) are live
    if (y,z) is live
        add block {x,y,z}
    else
        find and delete block {y,z,w}, add block {x,y,z}
```

```
Stinson()
    while |B| < n(n-1)/6
        do Switch()
    return (V,B)
```

Isomorphism

Using Invariants

invariants

- $SDS(G)$, sorted degree sequence

invariant inducing functions

- $D_{\triangle}(G, v)$, number of triangles in G passing through v.
- $D_{ngbd}(G, v)$, tuple/sequence representing the number of neighbors for each degree

general scheme

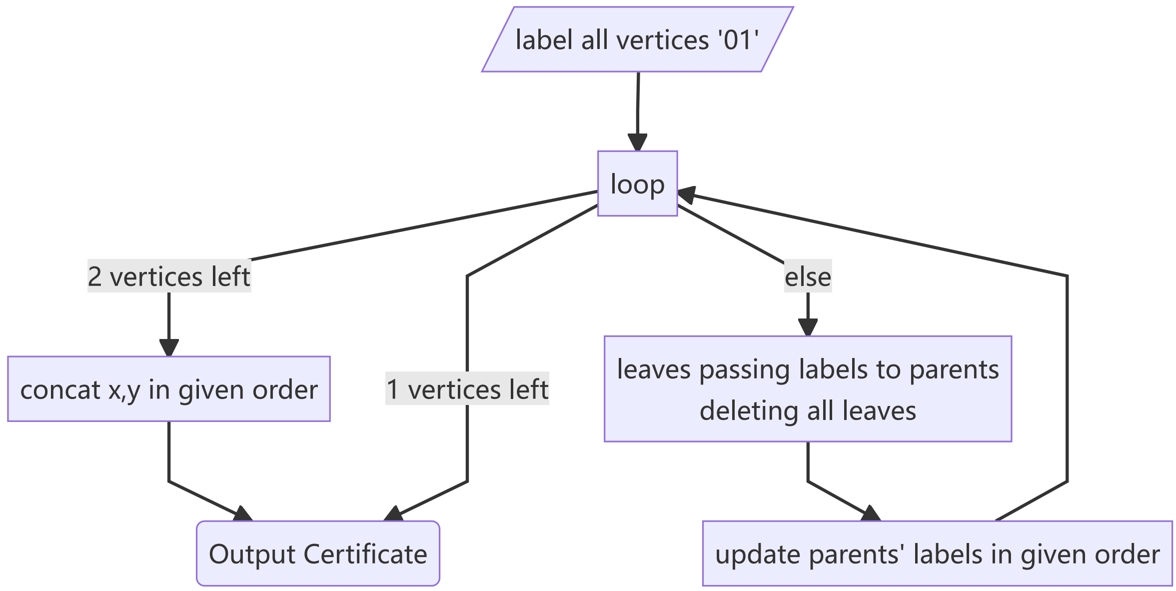
Initial vertex partition → refine partition by invariants → find isomorphisms using partition

Certificate (for tree)

$Cert: F \to \{0, 1\}^*$, F is a famlily of trees, $Cert(T)$=a balanced binary string of length $2|V|$

Encoding

```
Label all vertices with string 01.
While there are more than 2 vertices in G:
    for each non-leaft x of G do
    1. Let Y be the set of labels of the leaves adjacent to x and the label of
   x with initial 0 and trailing 1 deleted from x;
    2. Replace the label of x with the concatenation of the labels in Y ,
   sorted in increasing lexicographic order, with a 0 prepended and a 1
appended.
    3. Remove all leaves adjacent to x.
If there is only one vertex x left, report x’s label as the certificate.
If there are 2 vertices x and y left, concatenate x and y in increasing
lexicographic order, and report it as the certificate.
```

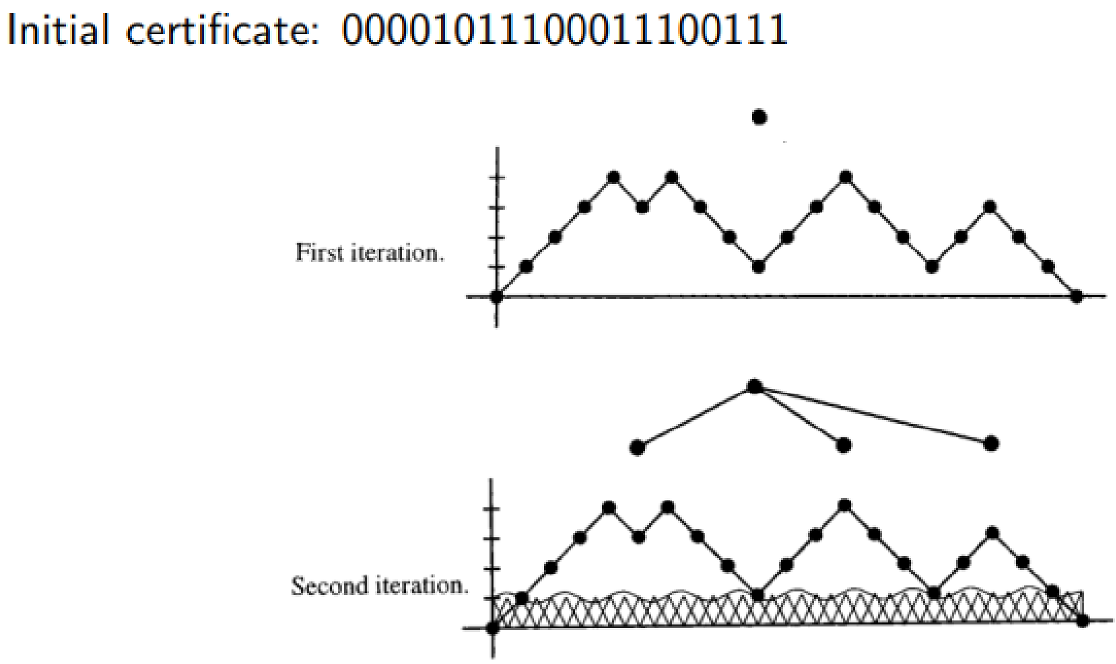


> in book: given order = increasing lexicographic order

Decoding

0=up, 1=down,

attach nodes correspond to mountain range at different see level (depth of tree).



Certificate (for general graph)

Cert1

- adjacency matrix
- adjacency number

  $Num(G) =$ binary string obtained by concatenate (upper half of) adjacency matrix column by column

  Example :

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \rightarrow 110101(\text{column by column})$$

- minimum num:

  $Cert_1 = min\{Num(\pi(G)) : \pi \in Sym(V)\}$

  > a certificate that difficult to compute. (NP-hard)

Cert2

Instead of going through all permutations, possible to go through a less class of permutations $\Pi(G)$, according to structure of a particular graph G.

> Recall: (equitable partition)
>
> graph $G = (V, E)$, adjacency of u, $N_G(u) = \{x \in V : \{u, x\} \in E\}$.
> partition P is equitable $\iff$ Given any i,
> $\forall u, v \in B[i], \forall j, |N_G(v) \cap B[j]| = |N_G(u) \cap B[j]|$.

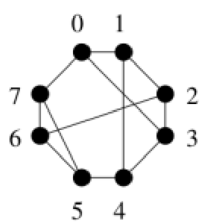
> Recall: (matrix associated to an equitable partition)
>
> Given equitable partition P with k blocks, matrix $M_P \in \mathbb{N}^{k \times k}$, $M_P[i, j] = |N_G(v) \cap P[j]|$, where $v \in P[i]$.

Similar to what is done in $Cert1$,

$Num_e(G, P) =$ binary string obtained by concatenate (upper half of) equitable matrix column by column.

Example : (the notation in the picture is a bit different)

$B = [\{0\}, \{2, 4\}, \{5, 6\}, \{7\}, \{1, 3\}]$ is an equitable partition w.r.t. $\mathcal{G}$:



$$M_B = \begin{bmatrix} 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \end{bmatrix}$$

and $Num(B) = [0, 0, 1, 1, 0, 1, 2, 2, 0, 0]$.

Lemma If P is a discrete partition,

then P corresponds to a permutation $\pi : P[i] = \{\pi[i]\}$, and $Num_e(G, P) = Num(\pi(G))$

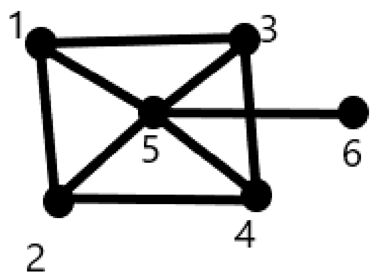
equitablization

Given a set of vertices S in graph G, define the equitablize function
$D_{G,S} : V \to \mathbb{N}$, $D_{G,S}(v) = |N_G(v) \cap S|$

Given an (ordered) partition P,

```
copy and push blocks of P into stack S
while S not empty
    B=S.pop()
    refine P by D_{G,B}
    push newly splitted blocks into S
```

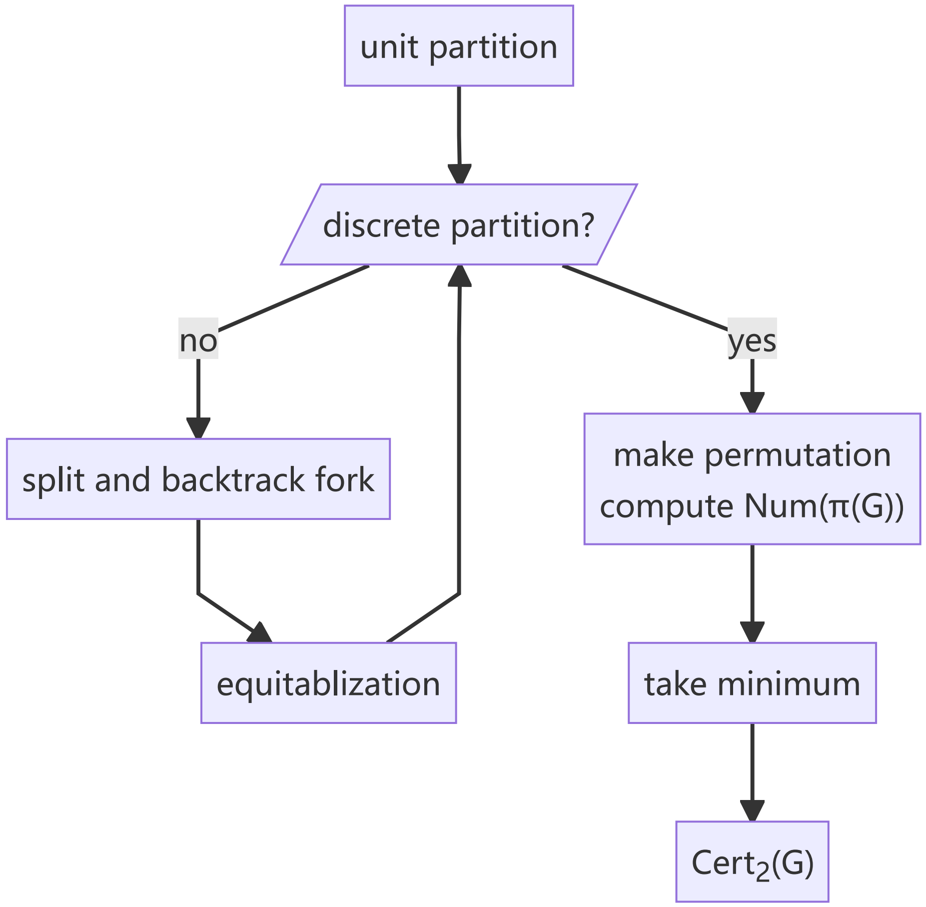
Example,



initial partition $P = \{\{1, 5\}, \{2, 3, 4, 6\}\}$ (NOT equitable), after refining,
$P = \{\{1\}, \{5\}, \{4\}, \{6\}, \{2, 3\}\}$

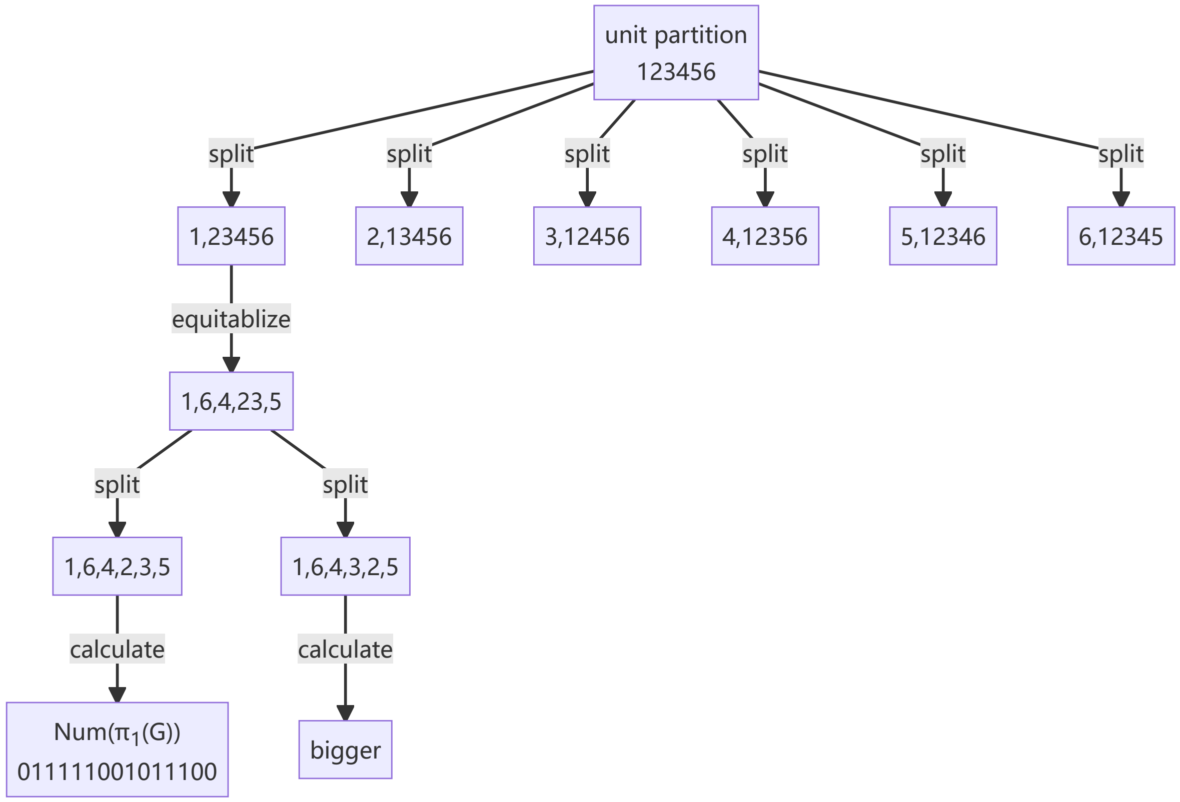
> Remark:
>
> If P is a unit partition ($P = \{V\}$), then refine P by $D_{G,V}$ is equivalent to refining by degree. (Since $D_{G,V}(v) =$ degree of v)

computing Cert2



> Remark:
>
> Actually, the comparation is done not when the whole certificate is computed, but every time the partial certificate is known. This can prune the search tree partially

Example :



> Remark:
>
> Notice when run this for complete graph, it will still go through $Sym(V)$.

***pruning with automorphisms**

Lemma $Num(\pi(G)) = Num(\pi'(G)) \implies \phi = \pi \circ \pi'^{-1} \text{is an automorphism}$

1. find automorphisms by Lemma
2. generate automorphism group using discovered automorphism as generators
3. use known automorphisms to prune the search

*Isomorphism of Others

colored graphs

preserving color

Set systems

Isomorphism of set system can be transformed to isomorphism of (bipartite) graph.

Given set system $S = (X, B)$, define bipartite graph $G = (V, E)$, where $V = X \cup B$, $E = \{(x, b) \in X \times B : x \in b\}$.

Theorem $S_1 \cong S_2 \implies G_1 \cong G_2$

> Remark:
>
> Converse is NOT true

Lemma
$S_1 = (X, B_1) \cong S_2 = (X, B_2) \iff G_1 \cong G_2 \text{ with respect to partition } P_1 = [X, B_1], P_2 = [X, B_2]$
And for isomorphism $\phi : G_1 \to G_2$ with respect to P_1, P_2, $\phi|_X$ is an isomorphism of S_1, S_2.

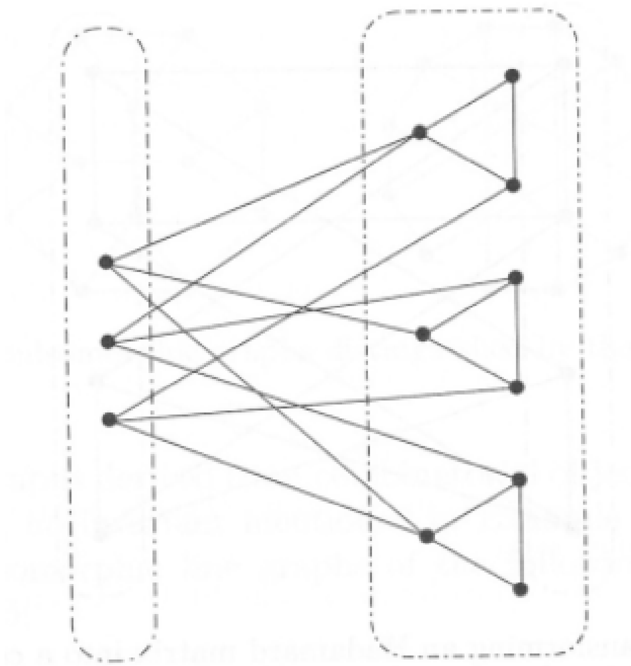
Codes

Isomorphism of codes can be transformed to isomorphism of colored graph.

Given q-ary code $C \subseteq \mathbb{Z}_q^n$, defined coloured graph $G = (V, E, f)$ with

- vertices, $V = C \cup A, \text{ where } A = (\{1, 2, \ldots, n\} \times \mathbb{Z}_q)$,
- edges,
  $E = \{\{x, (i, x[i])\} : x \in C, i \in \{1, 2, \ldots, n\}\} \cup \{\{(j, a), (j, b)\} : j \in \{1, 2, \ldots, n\}, a, b \in \mathbb{Z}_q\}$
- (vertex) colouring, $f(C) = \{0\}$, $f(A) = \{1\}$, that is, only 2 colors indicating code elements and positionals.

Example, code elements positionals



This graph can represent $\{000, 011, 220\} \cong \{000, 110, 022\} \cong \{111, 221, 100\}\ldots$

The End... Thanks for reading. Ray loves you ❤️.