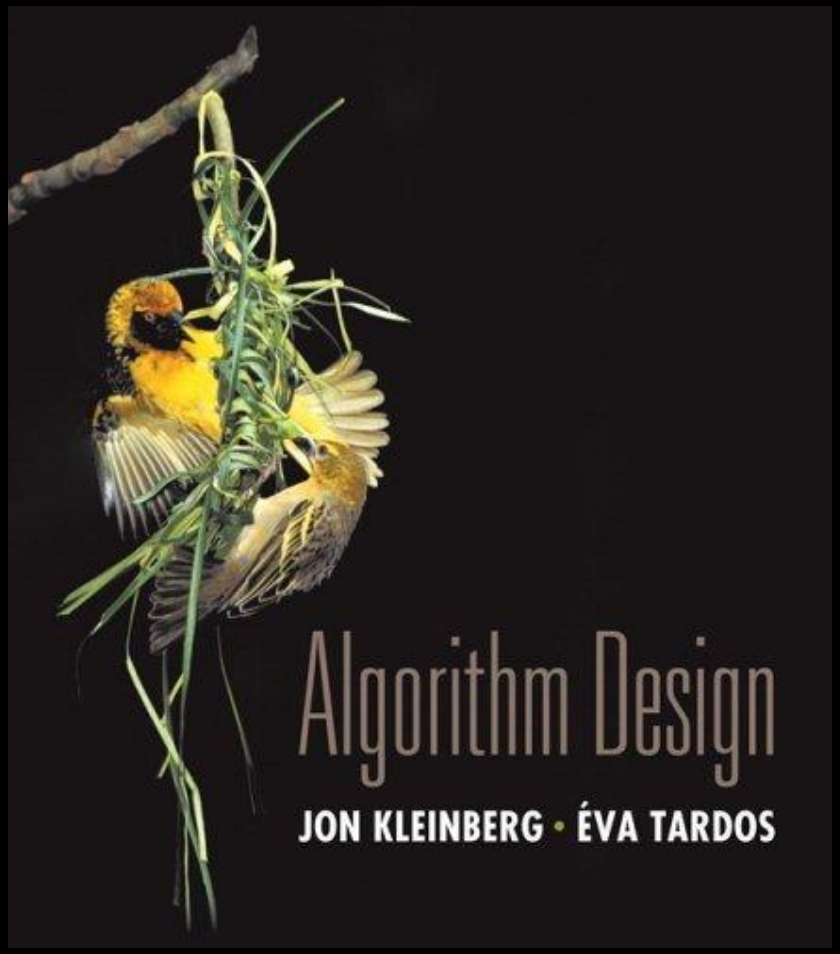


Chapter 13

Randomized Algorithms

Randomization

Algorithmic design patterns.

- Greed.
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Randomization.

in practice, access to a pseudo-random number generator

Randomization. Allow fair coin flip in unit time.

Why randomize? Can lead to simplest, fastest, or only known algorithm for a particular problem.

Ex. Symmetry breaking protocols, graph algorithms, quicksort, hashing, load balancing, Monte Carlo integration, cryptography.

Randomization

Conceitos Básicos: Seção 6.2 do Cormen edição 1.

1. Espaço amostral
2. Evento
3. Axiomas de Probabilidade
4. Probabilidade Condicional e Independencia
5. Union Bound

13.1 Verifying Polynomial Identities

Verifying Polynomial Identities

Given two polynomials F(x) and G(x) written in distinct forms, we want to know whether they are identical.

Is $x(x+1)(x-2)(x+3)(x-4)(x+5)(x-6) = x^6 - 7x^3 - 25$?

- We can convert both to their canonical form and then check the identity
  - It requires $O(d^2)$ time

Can we make this verification significantly faster?

Verifying Polynomial Identities

Assumptions

- Each multiplication and addition can be performed in constant time (more or less reasonable).
- We can pick an integer from a set with uniform probability

Algorithm

Select an integer r in the set {1,…,100d}
If F(r)=G(r) then
Output F and G are identical
Else
Output F and G are not identical

End

Running Time: O(d)

Verifying Polynomial Identities

Analysis

- If F(r)<>G(r) the algorithm outputs the correct answer.

- If F(r)=G(r) the algorithm may output the wrong answer. This only happens if r is a root of polynomial F-G. Since the degree of F-G is at most d, it follows from the funadamental theorem of algebra that F-G has at most d distinct roots.
  Thus, the probability of failure is at most d/ 100d = 1/100

Basics

Axioms of Probability

Definition 1.1 A probability space has three components

1. A sample space Ω, which is the set of all possible outcomes of the random process modeled by the probability space
2. A famility of sets F representing the allowable events, where each set in F is a subset of the sample space Ω
3. A probability function Pr : F → R satisfying Def 1,2

Definition 1.2 A probability function is any function Pr : F → R

1. For any event E, 0<= Pr(E)<=1
2. Pr(Ω)=1
3. For any finite or countably infinite sequence of pairwise mutually disjoint events $E_1, E_2, \ldots, E_k$

$$\Pr\left[\bigcup_i E_i\right] = \sum_i \Pr[E_i]$$

Verifying Polynomial Identities

- In the randomized algorithm, the sample space is {1,…,100d}. Each choice of r is a simple event.

- Each simple event has the same probability since the algorithm chooses r with uniform probability. It follows that the probability of a simple event is 1/100d

Basics

Lemma 1.1 For any two events E and F

$$\Pr(E \cup F) = \Pr(E) + \Pr(F) - \Pr(E \cap F)$$

Lemma 1.2 (Union Bound) For any finite or countably infinite sequence of pairwise mutually disjoint events

$$\Pr\left[\bigcup_i E_i\right] \leq \sum_i \Pr[E_i]$$

Verifying Polynomial Identities

Analysis

Let E be the event corresponding to algorithm failure.

$$E = \{ r \mid r \text{ in } \{1,\ldots,100d\} \text{ and } r \text{ is a root of } F\text{-}G \}$$

F-G has degree at most d ➔ the fundamental theorem of algbera assures that the number of roots of F-G is at most d . Thus,

$$\Pr(\text{algorithm fails}) = \Pr(E) <= d / 100d = 1/100$$

Verifying Polynomial Identities

- It is a bit unusual to design an algorithm that can return the wrong answer. What is the advantage?

- The algorithm provdes a trade-off between speed and correctness.

Verifying Polynomial Identities

Can we improve the probability of being correct?

- Enlarge the sample space: use {1,…,1000d} rather than {1,…100d}
  - Sometimes it is not possible to do it.

- Repeat the algorithm multiple times (try different values for r)
  - Sampling with replacement
  - Sampling without replacement

Verifying Polynomial Identities

Can we improve the probability of being correct?

- Enlarge the sample space: use {1,…,1000d} rather than {1,…100d}
  - Sometimes it is not possible to do it.

- Repeat the algorithm multiple times (try different values for r)
  - Sampling with replacement
  - Sampling without replacement

Verifying Polynomial Identities

Algorithm (Sampling with replacement)

```
For j=1...k
    Select an integer r in the set {1,...,100d}
    If F(r)<>G(r) then
            Output F and G are not identical
            Return
    End If
End For
Output F and are G identical
```

Running Time: O(kd)

Basics

Definition 1.3 (Independence) Two events E and F are independent if and only if

$$\Pr(E \cap F) = \Pr(E)\Pr(F)$$

Events E1,…,Ek are pairwise independent if and only if

$$\Pr(E_i \cap E_j) = \Pr(E_i)\Pr(E_j)$$

For every 1<=i<j<=k

More generally, events E1,…,Ek are mutually independent if and only if for any I subset of {1,…,k}

$$\Pr\left(\bigcap_{i\in I} E_i\right) = \prod_{i\in I} Pr\,(E_i)$$

Basics

Sample space S = { outcomes of flipping two fair coins}

S ={HH,HT, TH,TT}

E1= { firs coins is H}
E2 ={ coins are different}

E1 and E2 are independent because

$$Pr(E1 \cap E2) =1/4 \ =Pr\{E1\}Pr\{E2\}$$

Basics

Pairwise Independence does not imply mutual independence

Sample space S = { outcomes of flipping two fair coins}

S ={HH,HT, TH,TT}

E1= { first coins is H}
E2 ={ second coin is head}
E3 ={ two coins are different}

Pr{E1}=Pr{E2}=Pr{E3}=1/2
Pr{E1 \cap E2 } = Pr{E1 \cap E3 } = Pr{E2 \cap E3 } = $\frac{1}{4}$ (pairwise independence)

Pr{E1 \cap E2 \cap E3 } =0 <> 1/8 (not mutual independent)

Verifying Polynomial Identities

- Sample Space = {1,..100d} x {1...100d}x ....x {1,...,100d}
- E(i): Event that a root of F-G is selected in the i-th iteration
- Probability of failure

$$\Pr[\mathrm{E}(1) \cap E(2) \cap \cdots \cap E(k))] = \prod \Pr\big(E(i)\big) \leq \prod_{i=1}^{k} \frac{d}{100d} = \left(\frac{1}{100}\right)^{k}$$

Verifying Polynomial Identities

Algorithm (Sampling without replacement)

```
S ← {1,…,100d}
For j=1…k
    Select an integer r in the set S
    S← S-{r}
    If F(r)<>G(r) then
            Output 'F and G are not identical'
            Return
    End
End For
Output 'F and G are identical'
```

Running Time: O(kd)

Verifying Polynomial Identities

Definition 1.3 (Conditional Probability)
The conditional probability that E occurs given that F occurs is

$$\Pr(E|F) = \frac{Pr(E \cap F)}{\Pr(F)}$$

The probability is well defined if Pr(F)<>0

Basics

Lema 3: Seja $t_1, t_2, \ldots, t_k$ uma coleção de eventos. Temos que

$$\Pr\left[\bigcap_{i=1}^{k} t_i\right] = \prod_{i=1}^{k} \Pr\left[t_i \left| \bigcap_{j=1}^{i-1} t_j \right.\right]$$

Prova: Utilizar Indução no número de eventos

$$\Pr[t_2 | t_i] = \frac{\Pr[t_2 \cap t_1]}{\Pr[t_1]} \Rightarrow \Pr[t_1 \cap t_2] = \Pr[t_1] \cdot \Pr[t_2 | t_1] \text{ ok}$$

Verifying Polynomial Identities

- E(i): Event that a root of F-G is selected in the i-th iteration. Thus,

$$\Pr[\mathrm{E}(j)|E(1) \cap \cdots \cap E(j-1))] \leq \frac{d-(j-1)}{100d-(j-1)}$$

because the set S has 100d-(j-1) elements at the beginning of the j-th iterations and the number of roots of F-G in S is at most d-(j-1) since (j-1) have already been removed, for otherwise the algorithm would have already finished.

Thus,

$$\Pr[\mathrm{E}(1) \cap E(2) \cap \cdots \cap E(k))] = \prod_{i=1}^{k} \frac{d-(j-1)}{100d-(j-1)} \leq \left(\frac{1}{100}\right)^{k}$$

Verifying Polynomial Identities

Sampling with replacement x Sampling without replacement

- Sampling with replacement is usually simpler to analyse and yield to a reasonable

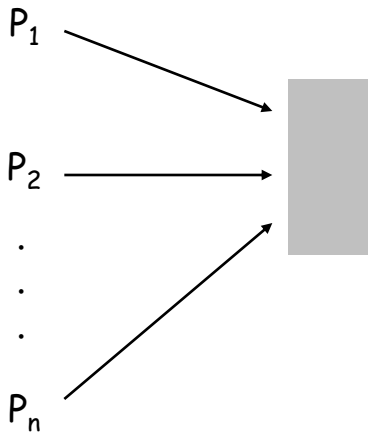
13.1 Contention Resolution

Contention Resolution in a Distributed System

Contention resolution. Given n processes P_1, …, P_n, each competing for access to a shared database. If two or more processes access the database simultaneously, all processes are locked out. Devise protocol to ensure all processes get through on a regular basis.

Restriction. Processes can't communicate.

Challenge. Need symmetry-breaking paradigm.



Contention Resolution: Randomized Protocol

Protocol. Each process requests access to the database at time t with probability p = 1/n.

Claim. Let S[i, t] = event that process i succeeds in accessing the database at time t. Then $1/(e \cdot n) \leq \Pr[S(i, t)] \leq 1/(2n)$.

Pf. By independence, $\Pr[S(i, t)] = p\,(1-p)^{n-1}$.

process i requests access

none of remaining n-1 processes request access

- Setting p = 1/n, we have $\Pr[S(i, t)] = 1/n\,(1 - 1/n)^{n-1}$. ▪

value that maximizes Pr[S(i, t)]

between 1/e and 1/2

Useful facts from calculus. As n increases from 2, the function:

- $(1 - 1/n)^n$ converges monotonically from 1/4 up to 1/e
- $(1 - 1/n)^{n-1}$ converges monotonically from 1/2 down to 1/e.

Contention Resolution: Randomized Protocol

Claim. The probability that process i fails to access the database in e×n rounds is at most 1/e. After e·n(c ln n) rounds, the probability is at most n^{-c}.

Pf. Let F[i, t] = event that process i fails to access database in rounds 1 through t. By independence and previous claim, we have $\Pr[F(i, t)] \leq (1 - 1/(en))^{t}$.

- Choose $t = \lceil e \cdot n \rceil$: $\Pr[F(i,t)] \leq \left(1-\frac{1}{en}\right)^{\lceil en \rceil} \leq \left(1-\frac{1}{en}\right)^{en} \leq \frac{1}{e}$
- Choose $t = \lceil e \cdot n \rceil \lceil c \ln n \rceil$: $\Pr[F(i,t)] \leq \left(\frac{1}{e}\right)^{c \ln n} = n^{-c}$

Contention Resolution: Randomized Protocol

Claim. The probability that all processes succeed within $2e \cdot n \ln n$ rounds is at least $1 - 1/n$.

Pf. Let F[t] = event that at least one of the n processes fails to access database in any of the rounds 1 through t.

$$\Pr[F[t]] = \Pr\left[\bigcup_{i=1}^{n} F[i,t]\right] \le \sum_{i=1}^{n} \Pr[F[i,t]] \le n\left(1-\tfrac{1}{en}\right)^{t}$$

union bound — previous slide

- Choosing $t = 2\lceil en \rceil \lceil c \ln n \rceil$ yields $\Pr[F[t]] \le n \cdot n^{-2} = 1/n$. ▪

Union bound. Given events $E_1, \ldots, E_n$, $\Pr\left[\bigcup_{i=1}^{n} E_i\right] \le \sum_{i=1}^{n} \Pr[E_i]$

13.2 Global Minimum Cut

Global Minimum Cut

Global min cut. Given a connected, undirected graph $G = (V, E)$ find a cut (A, B) of minimum cardinality.

Applications. Partitioning items in a database, identify clusters of related documents, network reliability, network design, circuit design, TSP solvers.

Network flow solution.

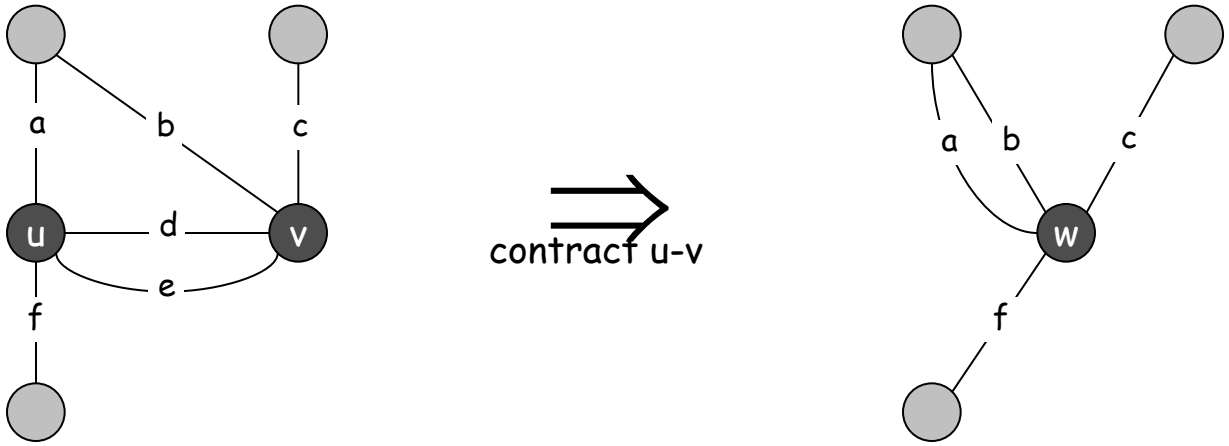
- Replace every edge (u, v) with two antiparallel edges (u, v) and (v, u).
- Pick some vertex s and compute min s-v cut separating s from each other vertex $v \in V$.

False intuition. Global min-cut is harder than min s-t cut.

Contraction Algorithm

Contraction algorithm. [Karger 1995]

- Pick an edge e = (u, v) uniformly at random.
- Contract edge e.
  - replace u and v by single new super-node w
  - preserve edges, updating endpoints of u and v to w
  - keep parallel edges, but delete self-loops
- Repeat until graph has just two nodes v_1 and v_2.
- Return the cut (all nodes that were contracted to form v_1).

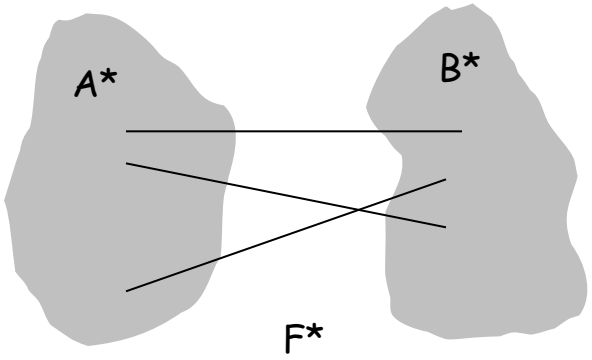


Contraction Algorithm

Claim. The contraction algorithm returns a min cut with prob $\geq 2/n^2$.

Pf. Fix some global min-cut (A^*, B^*) of G. Let F^* be edges with one endpoint in A^* and the other in B^*. Let $k = |F^*|$.

- In first step, algorithm contracts an edge in F^* probability $k / |E|$.
- Every node has degree $\geq k$ since otherwise (A^*, B^*) would not be min-cut. $\Rightarrow$ $|E| \geq \frac{1}{2}kn$.
- Thus, algorithm contracts an edge in F^* with probability $\leq 2/n$.



Contraction Algorithm

Claim. The contraction algorithm returns a min cut with prob $\geq 2/n^2$.

Pf. Fix some global min-cut (A^*, B^*) of G. Let F^* be edges with one endpoint in A^* and the other in B^*. Let $k = |F^*|$ = size of min cut.

- Let G' be graph after j iterations. There are $n' = n-j$ supernodes.
- Suppose no edge in F^* has been contracted. The min-cut in G' is still k.
- Since value of min-cut is k, $|E'| \geq \frac{1}{2}kn'$.
- Thus, algorithm contracts an edge in F^* with probability $\leq 2/n'$.

- Let E_j = event that an edge in F^* is not contracted in iteration j.

$$\begin{aligned}
\Pr[E_1 \cap E_2 \cdots \cap E_{n-2}] &= \Pr[E_1] \times \Pr[E_2 \mid E_1] \times \cdots \times \Pr[E_{n-2} \mid E_1 \cap E_2 \cdots \cap E_{n-3}] \\
&\geq \left(1-\tfrac{2}{n}\right)\left(1-\tfrac{2}{n-1}\right)\cdots\left(1-\tfrac{2}{4}\right)\left(1-\tfrac{2}{3}\right) \\
&= \left(\tfrac{n-2}{n}\right)\left(\tfrac{n-3}{n-1}\right)\cdots\left(\tfrac{2}{4}\right)\left(\tfrac{1}{3}\right) \\
&= \frac{2}{n(n-1)} \\
&\geq \frac{2}{n^2}
\end{aligned}$$

Contraction Algorithm

Amplification. To amplify the probability of success, run the contraction algorithm many times and return the best solution

Claim. If we repeat the contraction algorithm $n^2 \ln n$ times with independent random choices, the probability of failing to find the global min-cut is at most $1/n^2$.

Pf. By independence, the probability of failure is at most

$$\left(1-\frac{2}{n^2}\right)^{n^2 \ln n} = \left[\left(1-\frac{2}{n^2}\right)^{\frac{1}{2}n^2}\right]^{2\ln n} \leq \left(e^{-1}\right)^{2\ln n} = \frac{1}{n^2}$$

$(1 - 1/x)^x \leq 1/e$

Global Min Cut: Context

Improvement. [Karger-Stein 1996] $O(n^2 \log^3 n)$.

- Early iterations are less risky than later ones: probability of contracting an edge in min cut hits 50% when $n / \sqrt{2}$ nodes remain.
- Run contraction algorithm until $n / \sqrt{2}$ nodes remain.
- Run contraction algorithm twice on resulting graph, and return best of two cuts.

Global Min Cut:

FastCut($G=(V,E)$)

If $|V|<=6$

Calculate the min cut trough brute-force enumeration

Else

$t \leftarrow (1+ |V| / \sqrt{2})$

Using Algorithm Contract perform 2 independent sequences of contractions in G to obtain graphs H1 and H2, both with t vertices

$C1 \leftarrow$ FastCut(H1); $C2 \leftarrow$ FastCut(H2)

Return the smallest cut between C1 and C2

Global Min Cut:

Proposition. *A sequence of one contraction can be implemented in $O(n^2)$ time*

Proof.

Use the adjacency matrix reperesentation of a graph.

Keep a representative for each super node.

To contract na edge (i,j)

Find the representatives r(i) and r(j) of i and j, respectively

Global Min Cut:

Lemma 1. FastCut runs in $O(n^2 \log n)$

Proof. Let $T(n)$ be the time complexity of FastCut. Thus,

$$T(n) = O(n^2) + 2T(1 + n/\sqrt{2})$$

Solving the recurrence we establish the result.

Global Min Cut:

Lemma 2. FastCut succeeds in finding a min-cut with probability at least $c(1/\log n)$ for some constant c independent of n

Proof:

F: event that FastCut fails
F1 event that execution 1 fails
F2 event that execution 1 fails

FastCut fails if the two executions fail. Thus,

$$\Pr(F)=\Pr(F1)\Pr(F2)$$

S1(S2): event that execution 1(2) succeeds

$$\Pr(F2)=\Pr(F1)=1-\Pr(S1)=1-\Pr(S2)$$

Global Min Cut:

Lemma 2. FastCut succeeds in finding a min-cut with probability at least c(1/ log n) for some constant c independent of n

Proof:
Let P(t) be the probability that FastCut succeeds in finding a min-cut in a graph with t vertices.

A: event in which no edge in the min cut is contracted before the recursive call in execution 1
B: event in which a min cut is found in a graph with 1+t/ $\sqrt{2}$ vertexes

S1=A $\cap$ B

$$Pr(S1) = Pr(A \cap B) = Pr(A).Pr(B)$$

$Pr(A) = (t/\sqrt{2})\,(t/\sqrt{2}-1) / (t)(t-1) >= 1/2$ and

$Pr(B) = P(1+t/\sqrt{2})$

Global Min Cut:

Lemma 2. FastCut succeeds in finding a min-cut with probability at least c(1/ log n) for some constant c independent of n

Proof:

Thus,

$$Pr(S1) >= 1/2\ p(1+t/\sqrt{2})$$

We have that

$$Pr(F1)=Pr(F2)=1-Pr(S1) <=1 -1/2\ P(1+t/\sqrt{2})$$

$$Pr(F)<=(1 -1/2\ P(1+t/\sqrt{2}))^2$$

$$P(t)=1-Pr(F) >= 1- (1 -1/2\ P(1+t/\sqrt{2}))^2$$

Global Min Cut:

Lemma 2. FastCut succeeds in finding a min-cut with probability at least $c(1/\log n)$ for some constant c independent of n

Proof:

Thus, we have the following recursive equation

$$P(t) >= 1 - (1 - \tfrac{1}{2} P(1 + t/\sqrt{2})^2)$$

Solving this equation we get that $P(t) = 1/\log t$

Repeating the algorithm $\log^2 n$ we have success probability larger than (1/n)

Global Min Cut: Context

Remark. Overall running time is slow since we perform $\Theta(n^2 \log n)$ iterations and each takes $\Omega(m)$ time.

Improvement. [Karger-Stein 1996] $O(n^2 \log^3 n)$.

- Early iterations are less risky than later ones: probability of contracting an edge in min cut hits 50% when $n / \sqrt{2}$ nodes remain.
- Run contraction algorithm until $n / \sqrt{2}$ nodes remain.
- Run contraction algorithm twice on resulting graph, and return best of two cuts.

Extensions. Naturally generalizes to handle positive weights.

Best known. [Karger 2000] $O(m \log^3 n)$.

↖ faster than best known max flow algorithm or deterministic global min cut algorithm