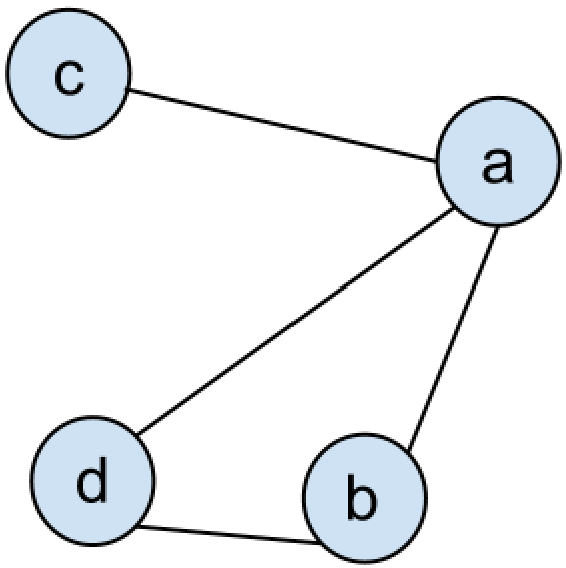
Sub-quadratic reductions

Detecting triangles = cycles of length 3

Input : G=(V,E)

Output: True if there is a triangle in G, False otherwise.

Example: (a,b,d) is a triangle in:



Using Matrix Multiplication

Input: Adjacency Matrix of G(V,E), M .

Recall: $M^{t}_{i,j} = ?$

Using Matrix Multiplication

Input: Adjacency Matrix of G(V,E), M .

Recall: $M^t_{i,j}$ = number of paths of length t from i to j.

Algorithm:

?

Using Matrix Multiplication

Input: Adjacency Matrix of G(V,E), M .

Recall: $M^t_{i,j}$ = number of paths of length t from i to j.

Algorithm:

- Compute M^3
- Check $M^3_{i,i}$ for all $1 \leq i \leq n$

  if one of them is not zero return True

  otherwise return False

Running time:

$2\ |V|^\omega + O(|V|) = O(|V|^\omega)$. Recall $\omega \leq 2.37$

Can we do better for sparse graphs?

Detecting triangles

Input : Adjacency List of G(V,E),

Output: True if there is a triangle in G, False otherwise.

Main idea of algorithm:

First we check for a triangle that has a node of degree $\leq \Delta$.

Then we look for a triangle with three nodes of degree $> \Delta$.

We can choose Δ as we please.

Algorithm

Let $\Delta := |E|^{(\omega-1)/(\omega+1)}$

Triangles with some node with degree $\leq \Delta$

For each edge (u,v) check if u or v has degree $\leq \Delta$

If so go through that node's neighbors w, and check if (u,v,w) is a triangle.

Time: ?

Algorithm

Let $\Delta := |E|^{(\omega-1)/(\omega+1)}$

Triangles with some node with degree $\leq \Delta$

For each edge (u,v) check if u or v has degree $\leq \Delta$

If so go through that node's neighbors w, and check if (u,v,w) is a triangle.

Time: $O(|E| \cdot \Delta)$

Triangles with every node with degree $> \Delta$

Sum of degrees = ?

Algorithm

Let $\Delta := |E|^{(\omega-1)/(\omega+1)}$

Triangles with some node with degree $\leq \Delta$

For each edge (u,v) check if u or v has degree $\leq \Delta$

If so go through that node's neighbors w, and check if (u,v,w) is a triangle.

Time: $O(|E| \cdot \Delta)$

Triangles with every node with degree $> \Delta$

Sum of degrees = 2|E|.

So there are ≤ ??????? nodes with degree $> \Delta$

Algorithm

Let $\Delta := |E|^{(\omega-1)/(\omega+1)}$

Triangles with some node with degree ≤ Δ

For each edge (u,v) check if u or v has degree ≤ Δ

If so go through that node's neighbors w, and check if (u,v,w) is a triangle.

Time: $O(|E| \cdot \Delta)$

Triangles with every node with degree > Δ

Sum of degrees = 2|E|.

So there are ≤ 2|E|/ Δ nodes with degree > Δ

Hence using matrix multiplication this takes ???

Algorithm

Let $\Delta := |E|^{(\omega-1)/(\omega+1)}$

Triangles with some node with degree ≤ Δ

For each edge (u,v) check if u or v has degree ≤ Δ

If so go through that node's neighbors w, and check if (u,v,w) is a triangle.

Time: $O(|E| \cdot \Delta)$

Triangles with every node with degree > Δ

Sum of degrees = 2|E|.

So there are ≤ 2|E|/ Δ nodes with degree > Δ

Hence using matrix multiplication this takes $O((|E|/ \Delta)^{\omega})$.

Overall: $O(|E| \Delta + (|E|/ \Delta)^{\omega}) =$

$= |E|^{1 + (\omega - 1)/(\omega + 1)} + |E|^{\omega(1 - (\omega - 1)/(\omega + 1))}$

$= |E|^{2\omega / (\omega + 1)} < |E|^{1.41}$ using $\omega < 2.38$

Recap: Can detect triangles in time $O(|E|^{2\omega / (\omega + 1)})$

So detecting triangles in time $|E|^{4/3}$ reduces to multiplying n x n matrices in time $O(n^2)$

Before trying to prove $\omega = 2$ you may want to try to detect triangles in time $|E|^{4/3}$

3SUM

Input: A set of numbers S, |S|=n.   Size of numbers = $n^{O(1)}$

Output: 1, if there are $a,b,c \in S$ such that $a+b+c=0$,
0, otherwise.

How long to solve 3SUM?

3SUM

Input: A set of numbers S, |S|=n.   Size of numbers = $n^{O(1)}$

Output: 1, if there are $a,b,c \in S$ such that $a+b+c=0$,

0, otherwise.

We can solve 3SUM in time $O(n^2)$.

It is believed that n^2 is optimal

Next: detecting triangles in time t

reduces to solving 3SUM in time O(t).

So, solving 3SUM in time $n^{1.4}$ would beat best-known triangle-detection algorithms (which run in $n^{1.41}$ time)

Next: detecting triangles in time t reduces
to solving 3SUM in time O(t).

- The reduction is randomized.

- We are going to give an algorithm R such that:
  if there is a triangle, R accepts with probability 1,
  otherwise R accepts with probability ≤ 1/100

- This gap can be amplified arbitrarily by ????

Next: detecting triangles in time t reduces
to solving 3SUM in time O(t).

- The reduction is randomized.

- We are going to give an algorithm R such that:
  if there is a triangle, R accepts with probability 1,
  otherwise R accepts with probability ≤ 1/100

- This gap can be amplified arbitrarily by repeating the algorithm a few times and taking Or

- It is possible to make R deterministic; we sketch that later

Detecting Triangles

Input: Adjacency list of graph $G(V,E)$. $|E|=m$.

Output: 1 if there is a triangle, 0 otherwise

Algorithm R:

1. Uniformly and independently assign a u-bit number to each node: $\forall\ a \in V,\ X_a \in \{0,1\}^u$

2. For each edge $(a,b) \in E$, compute $Y_{(a,b)}=(X_a - X_b)$ and $Y_{(b,a)}=(X_b - X_a)$.

3. Return answer of 3SUM on set $Y:=\{Y_{(a,b)},Y_{(b,a)} \mid (a,b) \in E\}$.

Analysis of R

- Suppose there is a triangle in G, say $\{(a,b), (c,b), (c,a)\}$.
- Note: graph is undirected, but the input is imposing an order which we eliminate by computing both $Y_{(a,b)}$, $Y_{(b,a)}$.
- The 3SUM instance contains numbers

$$Y_{(a,b)}+Y_{(b,c)}+Y_{(c,a)} = (X_a - X_b) + (X_c - X_b) + (X_c - X_a)$$

What is the probability that the sum will be 0?

Analysis of R

- Suppose there is a triangle in G, say {(a,b), (c,b), (c,a)}.
- Note: graph is undirected, but the input is imposing an order which we eliminate by computing both $Y_{(a,b)}$, $Y_{(b,a)}$.
- The 3SUM instance contains numbers

$$Y_{(a,b)}+Y_{(b,c)}+Y_{(c,a)} = (X_a - X_b) + (X_c - X_b) + (X_c - X_a)$$

Pr[R(G)=1] =1

That is, if there is a triangle we catch it.

Analysis of R

- Assume G does not have triangle

We want to show $\Pr[R(G)=1] < 1/100$

S_0 := some 3 numbers in Y sum to zero.

- $S(e_1,e_2,e_3)$:= the values corresponding to three distinct edges $e_1=(a_1,b_1),e_2=(a_2,b_2),e_3=(a_3,b_3)$, sum to zero.

$\Pr[R(G)=1] = \Pr[S_0]=$

$$= \Pr[\text{exists } e_1,e_2,e_3 \in E,\ S(e_1,e_2,e_3)] \le \sum_{e_1,e_2,e_3} \Pr[S(e_1,e_2,e_3)]$$

$\Pr[S(e_1,e_2,e_3)] = \Pr[Y_{e1}+Y_{e2}+Y_{e3}=0]$

$= \Pr[X_{a1}+X_{a2}+X_{a3}=X_{b1}+X_{b2}+X_{b3}]$

There are no triangles in G ➔ some node appears only once

➔ one of the variables in $X_{a1}+X_{a2}+X_{a3}=X_{b1}+X_{b2}+X_{b3}$ appears only once. Let that variable be X_{a1}

For any fixed choices of the other variables, there is ≤ 1 choice for X_{a1} that satisfies the equation.

So $\Pr[S(e_1,e_2,e_3)] \le 1/2^u$

Hence, $\Pr[S_0] \le \sum \Pr[S(e_1,e_2,e_3)] \le |E|^3 / 2^u$

Setting u = 3 log |E| + 7 we have $\Pr[R(G)=1] \le 1/100$

Making the reduction deterministic.

Need to construct m numbers X_a such that

$(X_a - X_b) + (X_c - X_d) + (X_e - X_f) = 0$

➔ each number is repeated twice, with opposite signs

This guarantees that they correspond to a triangle.

Note, numbers must have magnitude ≤ poly(m)
Otherwise, both easy and uninteresting (exercise: why?)

We are going to sketch the idea and leave details to exercises

Need to construct m numbers X_a such that

$(X_a - X_b) + (X_c - X_d) + (X_e - X_f) = 0$

→ each number is repeated twice, with opposite signs

- Construct m sets $S_1, S_2, \ldots, S_m \subseteq \{1, 2, \ldots, u \log m\}$:

  $|S_a| = c \log m, \forall a$

  $|S_a \cap S_b| < (c/5) \log m, \forall a \neq b,$

  for some constants u and c

- Then set X_a to be the number with u digits in base 10, where digit i is 1 if $i \in S_a$, 0 otherwise

- Exercise: Show that such X_a satisfy above (hint: no carry)
- Exercise: Construct such sets in time exponential in m (can be made time O(m), which is what is needed)

Recall

All-pairs shortest paths

Dynamic programming approach:

$d_{i,j}^{(m)}$ = shortest paths of lengths ≤ m

$d_{i,j}^{(m)} = \min_k \{ d_{i,k}^{(m-1)} + w(k,j) \}$

(Includes k = j, w(j,j) = 0)

Compute |V| x |V| matrix $d^{(m)}$ from $d^{(m-1)}$ in time $|V|^3$.

➔ $d^{|V|}$ computables in time $|V|^4$

How to speed up?

Recall

All-pairs shortest paths

Note:

$d_{i,j}^{(m)} = \min_k \{ d_{i,k}^{(m-1)} + w(k,j) \}$

Is just like matrix multiplication: $d^{(m)} = d^{(m-1)} W$,

except $+ \rightarrow \min$

$\quad\quad x \rightarrow +$

Like matrix multiplication, this is associative. So, instead of doing $d^{|V|} = (\ldots)W)W)W$ can do ?

Recall

All-pairs shortest paths

Note:

$d_{i,j}^{(m)} = \min_k \{ d_{i,k}^{(m-1)} + w(k,j) \}$

Is just like matrix multiplication: $d^{(m)} = d^{(m-1)} W$,
except $+ \rightarrow \min$
$\quad x \rightarrow +$

Like matrix multiplication, this is associative. So,
instead of doing $d^{|V|} = (\ldots)W)W)W$ can do repeated squaring:

Compute $d^{(2)} = W^2$
$\quad d^{(4)} = d^{(2)} \times d^{(2)} = W^2 \times W^2$
$\quad d^{(8)} = d^{(4)} \times d^{(4)}$
$\quad \ldots$

To get $d^{|V|}$ need ?

Recall

All-pairs shortest paths

Note:

$d_{i,j}^{(m)} = \min_k \{ d_{i,k}^{(m-1)} + w(k,j) \}$

Is just like matrix multiplication: $d^{(m)} = d^{(m-1)} W$,
except $+ \longrightarrow \min$
$\quad x \longrightarrow +$

Like matrix multiplication, this is associative. So,
instead of doing $d^{|V|} = (\ldots)W)W)W$ can do repeated squaring:

Compute $d^{(2)} = W^2$

$d^{(4)} = d^{(2)} \times d^{(2)} = W^2 \times W^2$

$d^{(8)} = d^{(4)} \times d^{(4)}$

...

To get $d^{|V|}$ need log |V| multiplications only ➔ $|V|^3 \log |V|$ time

- We used (Min,+) Matrix product in time t to solve APSP in time t log |V|

In particular, computing APSP in time $|V|^2$ log |V| reduces to computing (Min,+) Matrix product in time $|V|^2$

- Next: Use APSP to solve (Min, +) Matrix product.

(Min, +) Matrix product:
Input: Matrices A_{nxn} and B_{nxn}.
Output: C_{nxn} such that $C_{i,j} = \min_k \{A_{i,k} + B_{k,j}\}$.

We need to convert A and B to an instance of APSP.

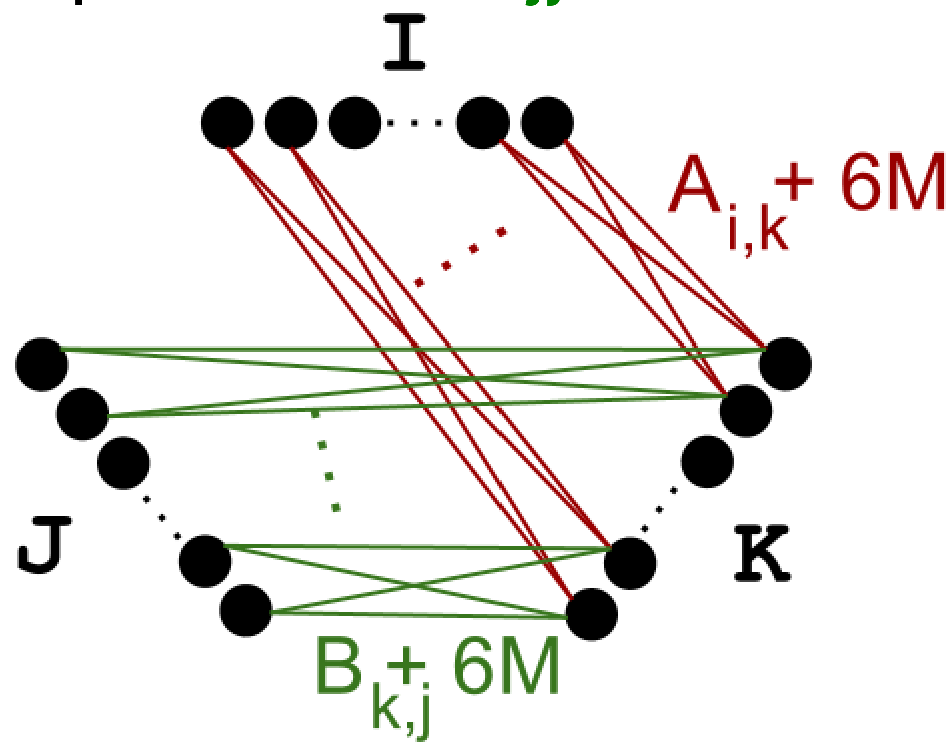
1. Let entries of A and B $\in$ [-M,M]
create a tripartite graph G (I,J,K , E), with n nodes in each part I, J and K,

$\forall$ i $\in$ I , k $\in$ k , (i,j) $\in$ E and $w(i,k)= A_{i,k} + 6M$.

$\forall$ k $\in$ k, j $\in$ J , (j,k) $\in$ E and $w(k,j)= B_{k,j} + 6M$.

2.Run the algorithm for APSP on G.
3.set $C_{i,j}$:= {length of the shortest path from i to j}-12M.

Why ?



Note:

Any path of length ≥ 3 weights $\geq 3(-M + 6M) \geq 15M$,

Any path of length ≤ 2 weights $\leq 2(M + 6M) \leq 14M$.

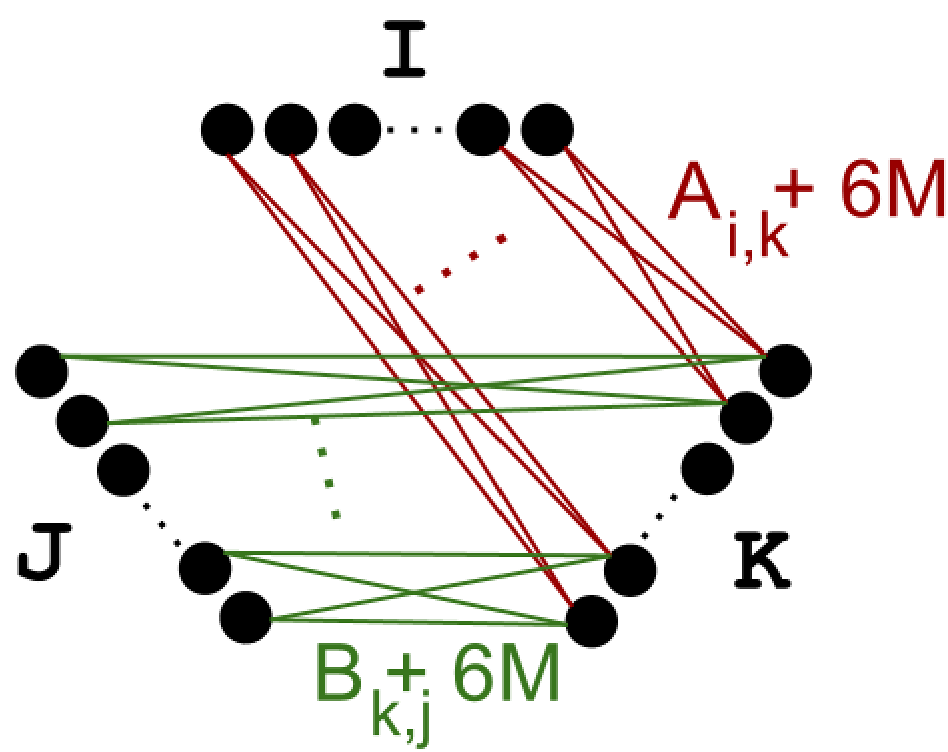
$\forall\ i \in I,\ j \in J$ there is a path of length 2 from i to j.

Therefore the shortest path from i to j is:

$\min_k \{w(i,k)+w(k,j)\}$,

$= \min_k \{A_{i,k}+6M+B_{k,j}+6M\}$,

$= \min_k \{A_{i,k}+B_{k,j}\}+12M$



- Running time:

Creating graph G : Takes $O(n^2)$

So we compute (Min,+) Matrix product of nxn matrices in time $O(n^2)$ + APSP-TIME(3n).

- Putting both reductions together:

APSP and (Min,+) Matrix product are basically the same problem.

Either both of them can be solved in time $n^{3-\varepsilon}$, or neither can