Subcubic Equivalence of Triangle Detection and Matrix Multiplication

Bahar Qarabaqi and Maziar Gomrokchi

April 29, 2011

1 Introduction

An algorithm on $n \times n$ matrix with the entries in $[-M, M]$ has a truly subcubic run-time order if it runs in the order of $O(n^{3-\delta}.Poly(logM))$ for some $\delta > 0$. This notion applies to the graphs with n nodes and edge-weights in $[-M, M]$.

The following problems either all have truly subcubic algorithms, or none of them do:

- The all-pairs shortest paths problem on weighted digraphs (APSP).
- Detecting if a weighted graph has a triangle of negative total edge weight.
- Listing up to $n_2.99$ negative triangles in an edge-weighted graph.
- Finding a minimum weight cycle in a graph of non-negative edge weights.
- The replacement paths problem on weighted digraphs.
- Finding the second shortest simple path between two nodes in a weighted digraph.
- Checking whether a given matrix de?nes a metric.
- Verifying the correctness of a matrix product over the (min, +)-semiring.

In this report we only prove the subcubic-equivalence between Detecting Negative Triangle in an edge-weighted graph and matrix product over the (min, +)-semiring problems. We define the notion of subcubic reducibility as follows:

Definition 1. Let A and B be computational problems with the same input size n. Then we say there is a subcubic reduction from A to B ($A \leq_3 B$), if there exist an algorithm F with the oracle access to B such that for every $\delta > 0$ and $\epsilon > 0$ satisfies the following three properties:

- F runs in $O(n^{3-\delta})$ time with input size of n.

- For very instance x of A with size n, let n_i be the size of the ith oracle call to B in $F(x)$. Then $\Sigma_i\ {n_i}^{3-\epsilon} \leq n^{3-\delta}$

Therefore, we say A and B are subcubic-equivalent if and only if $A \leq_3 B$ and $B \leq_3 A$.

2 Intuition and Preliminaries

2.1 Intuition

The key observation is the counterintuitive result that subcubic algorithms for certain triangle detection problems can be used to obtain subcubic matrix products in many forms, including products that are not known to be subcubic. We first explain two main intuitions for why fast triangle detection should not imply fast matrix multiplications and then we talk about our approach. First that triangle detection returns one bit, while fast matrix multiplication returns n^2 bits. Second, fast matrix multiplication algorithm can determine for all edges if there is a triangle using the edge, while triangle detection only determines if some edge is in a triangle. Given these two intuitions about quantifiers, it looks unlikely that the universally quantified problem could be efficiently reduced to the existentially quantified problem. So there appears to be strong intuition for why such a reduction would not be possible.

2.2 Preliminaries

We start with a general definition encompassing all algebraic structures. Let R be a finite set. A $(min, \odot)$ structure over R is defined by a binary operation $\odot : R \times R \rightarrow Z \cup \{-\infty, \infty\}$. We use the variable R to refer to a $(min, \odot)$ structure. We say a $(min, \odot)$ structure is extended if $R \subset Z$ and R contains elements ϵ_0 and ϵ_1 such that for all $x \in R$, $x.\epsilon_0 = \epsilon_0.x = \infty$ and $\epsilon_1.x = x$ for all $x \in R$. That is, ϵ_0 is a type of annihilator, and ϵ_1 is a left identity. We use the variable $\bar{R}$ to refer to an extended structure. The elements ϵ_0 and ϵ_1 allow us to define (for every n) an $n \times n$ identity matrix I_n and a $n \times n$ zero matrix Z_n over $\bar{R}$. More precisely, $I_n[i, j] = \epsilon_0$ for all $i \neq j$, $I_n[i, i] = \epsilon_1$, and $Z_n[i, j] = \epsilon_0$ for all i, j . We shall omit the subscripts of I_n and Z_n when the dimension is clear.

Matrix Products Over Structures. The matrix product of two $n \times n$ matrices over R is:

$$(A \odot B)[i, j] = \min_{k \in [n]}(A[i, k] \odot B[k, j]).$$

Informal Statement of Theorems 1 **and** 2: Let $\bar{R}$ be an extended $(min, \odot)$ structure. The following problems over $\bar{R}$ either all have truly subcubic algorithms, or none of them do:

- **Negative Triangle Detection.** Given an n-node graph with weight function $w : V \times V \rightarrow Z$, find nodes i, j, k such that $w(i, j) \in Z$, $w(i, k) \in Z$, $w(k, j) \in Z$, and $(w(i, j) \odot w(k, j)) + w(i, j) < 0$.

- **Matrix Product.** Given two $n \times n$ matrices A, B with entries from Z, compute the product of A and B over $\bar{R}$.

The abovementioned reductions crucially rely on the fact that the addition operation in a $(min, \odot)$ structure is a minimum.

3 Negative Triangle Detection is subcubic-equivalent with Matrix Product

We first show that matrix product can solve the negative triangle problem over any extended structure $\bar{R}$ in the same asymptotic runtime and then we show that negative triangle problem over any extended structure $\bar{R}$ can solve matrix product in the same asymptotic runtime. For two problems A and B, we write $A \leq_3 B$ to express that there is a subcubic reduction from A to B.

3.1 Negative Triangle Detection Implies Matrix Product

Theorem 1. ***Negative Triangle Over*** $\bar{R} \leq_3$ ***Matrix Product Over*** $\bar{R}$**.** *Suppose matrix product over $\bar{R}$ can be done in time $T(n)$, then the negative triangle problem for graphs over $\bar{R}$ can be solved in $O(T(n))$ time.*

Proof. Algorithm:

A tripartite graph $G = \{I \cup J \cup K, E\}$ is given by the negative triangle detection over $\bar{R}$, where I, J, K are three disjoint sets of nodes in the graph and E is the set of edges. Given the weight function $w : V \times V \to Z \cup \{\infty, -\infty\}$. We first construct A, B, C matrices as follows:

- For each edge $(i, j) \in I \times J \cap E$ set $C[i, j] = w(i, j)$.
- For each edge $(j, k) \in J \times K \cap E$ set $B[j, k] = w(j, k)$.
- For each edge $(i, k) \in I \times K \cap E$ set $A[i, K] = w(i, k)$.
- When there is no edge in the graph, the corresponding matrix entry in A or B becomes ε_0 and in C it becomes ∞.

We know that nodes i, j, k from graph G form a negative triangle if and only if $(w(i, k) \odot w(k, j)) < w(i, k)$. Knowing this, the problem becomes to determine whether there are $i, j, k \in [n]$ so that $A[i, k] \odot B[k, j] < C[i, j]$. Let A' be the $n \times 2n$ matrix obtained by concatenating A to the left of the $n \times n$ identity matrix I. Let B' be the $2n \times n$ matrix obtained by concatenating B on top of C. Then if we run matrix product algorithm on matrices $A' and B'$, this is the same as componentwise minimum of $A \odot B$ and C. We can expand $A', B' and\ C$ to $2n \times 2n$ matrices by concatenating and $n \times 2n$ matrix of ϵ_0s to

the button of A' , all ϵ_0s $2n \times n$ matrix to the right of B' and n columns of all ϵ_0s and n rows of all ϵ_0s to the right and bottom of C.

Now given A' and B', we run the matrix product algorithm (with the subcubic runtime order) and it returns the matrix C'. Since:

$$\min_k(A'[i,k] \odot B'[k,j]) = \min\{C[i,j], \min_k(A[i,k], B[k,j])\} \leq C[i,j]$$

and given that $\min_k(A'[i,k] \odot B'[k,j]) \neq C[i,j]$, then we can run a componentwise comparison (with the order of $O(n^2)$) between C and C', if it returns false then there must exist a $k \in [n]$ such that $A[i,k] \odot B[k,j] < C[i,j]$ and therefore a negative triangle over $\bar{R}$. But if for all i, j, $C'[i,j] = C[i,j]$ then based on the right hand side of abovementioned equation, for all i, j we have $\min_k(A[i,k] \odot B[k,j]) \geq C[i,j]$ and therefore no negative triangle.

Correctness: The algorithm is correct by construction.

Runtime: Based on the abovementioned algorithm, construction of matrices A, B and C is in the order of $O(n^2)$, the componentwise comparison between C and C' is in the order of $O(n^2)$ and since we only have one oracle call to the matrix multiplication algorithm with the order of $\mathrm{O}(n^{3-\delta}.Poly(logM))$, therefore the negative triangle detection algorithm is in the order of $\mathrm{O}(n^{3-\delta}.Poly(logM))$.

□

3.2 Matrix product over R $\leq_3$ Negative triangle over R

Before we prove the other direction, we state and prove two lemmas that will be later used in the proof of Theorem 2.

Lemma 1. *If the negative triangle detection over R on a tripartite graph G can be done in $T(n)$ time, then there is an algorithm that returns a negative triangle over R in G in time $O(T(n))$ if one exists. Assume that $T(n) = nf(n)$ where $f(n)$ is a non-decreasing function.*

Proof. Algorithm: In algorithm 1, NTD[G] is a procedure that takes graph G as input and returns true if exists a negative triangle in G and false otherwise. Also, $E_{I_iJ_jK_k}$ is a subset of the edges in G that contains only the edges between the nodes in I_i, J_j and K_k.

Correctness: The procedure is recursive: First splits each of the three parts I, J and K into two parts of almost equal size. Then iterates through all 8 possible ways to choose a triple of these subparts and runs the negative triangle detection algorithm on each. If one returns 'true', skips running the detection algorithm on the other triples and recursively calls the current algorithm on that triple. If none returns 'true', there is no negative triangle in G. The procedure continues until each part in the triple that returns 'true' has only one node. Those nodes are the nodes of a negative triangle.

Run time: Let $T'(n)$ be the run time of the algorithm. From the algorithm, we know that:

Algorithm 1 Negative Triangle Finding

NTF[G]

1: **if** G has only 1 node in each part I, J, K **then**
2: **if** NTD[G] = true **then**
3: **return** G
4: **else**
5: **return** *null*
6: **end if**
7: **end if**
8: Split I, J, K each into 2 parts $I_i, J_j, K_k : i, j, k \in \{1, 2\}$ of almost equal size
9: **for all** 8 combinations of $I_i, J_j, K_k : i, j, k \in \{1, 2\}$ **do**
10: $G' := (I_i \cup J_j \cup K_k, E_{I_i J_j K_k})$
11: **if** NTD[G'] = true **then**
12: **goto** 16
13: **end if**
14: **end for**
15: **return** null
16: NTF[G']

$$T'(n) = 8T(n) + T'(n/2)$$
$$T'(1) = O(1)$$

$$\begin{aligned} T'(n) &= 8T(n) + 8T(n/2) + 8T(n/4) + ... + 8T(1) + T'(1) \\ &\leq 8cnf(n)(1 + 1/2 + 1/4 + ... + 1/n) + O(1) \\ &\leq 16cnf(n) + O(1) \\ &= O(nf(n)) = O(T(n)) \end{aligned}$$

Note that a special case is $T(n) = O(n^{3-\delta})$. □

Lemma 2. *If the negative triangle finding over R on a tripartite graph G can be done in $T(n)$ time, then there is an algorithm that lists all IJ-disjoint negative triangles over R in G in $O(T(n^{1/3})n^2)$ time.*

Proof. Algorithm:

Correctness: The algorithm splits each part I, J and K into n^a parts, each having at most $\lceil n^{1-a} \rceil$ nodes. Parameter a will be set later. It iterates through all n^{3a} possible ways to choose a triple of parts and considers the subgraph $G' \subset G$ that is generated from the nodes in the triple and the edges between its nodes. Then repeatedly runs the algorithm from Lemma 1 on G'; each time a negative triangle is found, adds it to the list of triangles and removes edge (i, j) from G' and G. The procedure continues until

Algorithm 2 Negative Triangle Listing

NTL[G]

1: Split I, J, K each into n^a parts $I_i, J_j, K_k : i, j, k \in \{1, ..., n^a\}$ of almost equal size
2: **for all** n^{3a} combinations of $I_i, J_j, K_k : i, j, k \in \{1, ..., n^a\}$ **do**
3: $G' := (I_i \cup J_j \cup K_k, E_{I_i J_j K_k})$
4: **while** NTF[G'] returns a negative triangle t=$(\hat{i}, \hat{j}, \hat{k})$ **do**
5: add t to the list L
6: remove $(\hat{i}, \hat{j})$ from G' and G
7: **end while**
8: **end for**
9: **return** L

G' contains no more negative triangle. The same procedure is repeated for all triples of parts. Note that we are looking for IJ-disjoint negative triangles. If some edge (i, j) is in a negative triangle, it is found when the negative triangle finding algorithm is performed on one of the triple parts that contains a negative triangle including that edge. When the triangle is detected, the edge is removed from the graph and hence, it is not in any triangle anymore.

Run time: Let $t_{I_i J_j K_k}$ be the number of triangles that are found in the set of IJ-disjoint negative triangles when (I_i, J_j, K_k) is processed. Let $T(n)$ be the run time of the negative triangle finding algorithm in G. According to the algorithm, the run time is

$$T'(n) = O(\sum_{n^{3a} \text{ triples } I_i, J_j, K_k} (t_{I_i J_j K_k} T(n^{1-a}) + T(n^{1-a})))$$

Note that $t_{I_i J_j K_k} T(n^{1-a})$ is for finding the $t_{I_i J_j K_k}$ negative triangles and the second $T(n^{1-a})$ is to make sure that there is no more negative triangle left. Also note that there are at most n^2 IJ-disjoint negative triangles in the graph and therefore, the sum of all $t_{I_i J_j K_k}$ is at most n^2. Hence,

$$\begin{aligned} T'(n) &= O(\sum_{n^{3a} \text{ triples } I_i, J_j, K_k} (t_{I_i J_j K_k} T(n^{1-a}) + T(n^{1-a}))) \\ &\leq cT(n^{1-a}) \sum_{n^{3a} \text{ triples } I_i, J_j, K_k} (t_{I_i J_j K_k} + 1) \\ &\leq cT(n^{1-a})(n^2 + n^{3a}) \end{aligned}$$

Now set $a := 2/3$, the run time becomes $O(T(n^{1/3})n^2)$. An immediate corollary is that if $T(n)$ is subcubic, then $T'(n)$ is also subcubic.

□

Theorem 2. *If the negative triangle detection over R in an n-node graph can be solved in $T(n)$ time, then the product of two $n \times n$ matrices over R can be performed in $O(n^2 T(n^{1/3}) logW)$ time, where W is the largest absolute value in the output.*

Proof. Algorithm: Let A and B be the given matrices in algorithm 3. Suppose the integers in the output $A \odot B$ lie in $[-W, W]$.

Algorithm 3 Matrix Product

MP$[A, B, W]$

1: generate $n \times n$ matrices U and L
2: $U[i,j] \leftarrow W+1$, $L[i,j] \leftarrow -W : \forall i,j \in \{1,...,n\}$
3: generate a complete tripartite graph G with weight function $w(.)$
4: $w(i,k) \leftarrow A[i,k]$, $w(k,j) \leftarrow B[k,j]$
5: **repeat**
6: $\quad w(i,j) \leftarrow \left\lceil \frac{U[i,j]+L(i,j)}{2} \right\rceil$
7: $\quad$ List $L_{NT} \leftarrow$ NTL$[G]$
8: $\quad$ **for all** $i,j \in \{1,...,n\}$ **do**
9: $\quad\quad$ **if** (i,j) appears in a negative triangle in L_{NT} **then**
10: $\quad\quad\quad U[i,j] \leftarrow w(i,j)$
11: $\quad\quad$ **else**
12: $\quad\quad\quad L[i,j] \leftarrow w(i,j)$
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: **until** $U[i,j] = L[i,j]+1 : \forall i,j \in \{1,...,n\}$
16: **return** L

Correctness: The algorithm correctly finds the matrix $C = A \odot B$ by performing a binary search on $[-W, W]$ for each entry in C. $U[i,j]$ determines the exclusive upper bound for $C[i,j]$ and $L[i,j]$ determines the inclusive lower bound. Note that since $C[i,j] = \min_k\{A[i,k] + B[k,j]\}$, it is equal to $A[i,k] + B[k,j]$ for some k. Every time a negative triangle with nodes i and j is detected, it means that there exists some k such that $A[i,k] + B[k,j] < w(i,j)$ and hence, $C[i,j]$ is less than $w(i,j)$. Therefore, we set the upper bound for i,j to the current $w(i,j)$. But if no negative triangle is detected for nodes i,j we must update the lower bound.

Run time: In each iteration, we need to find the list of IJ-disjoint negative triangles. Applying Lemma 2, this can be done in $O(T(n^{1/3})n^2)$. Due to the binary search on $[-W, W]$, the total number of iterations is $\log W$ and hence, the algorithm can be done in $O(n^2T(n^{1/3})logW)$.

□

References

[1] V. Vassilevska Williams and R. Williams, *Subcubic Equivalences Between Path, Matrix, and Triangle Problems*. In 51st IEEE Symposium on Foundations of Computer Science, 2010.

[2] V. Vassilevska Williams and R. Williams, *Triangle Detection Versus Matrix Multiplication: A Study of Truly Subcubic Reducibility*. 2010.