Finding the largest triangle in a graph in expected quadratic time

Giuseppe Lancia*and Paolo Vidoni†

Abstract

Finding the largest triangle in an n-nodes edge-weighted graph belongs to a set of problems all equivalent under subcubic reductions. Namely, a truly subcubic algorithm for any one of them would imply that they are all subcubic. A recent strong conjecture states that none of them can be solved in less than $\Theta(n^3)$ time, but this negative result does not rule out the possibility of algorithms with average, rather than worst-case, subcubic running time. Indeed, in this work we describe the first truly-subcubic average complexity procedure for this problem. For graphs whose edge lengths are uniformly distributed in $[0, 1]$ our procedure finds the largest triangle in average quadratic time, which is the best possible complexity of any algorithm for this problem. We also give empirical evidence that the quadratic average complexity holds for many other random distributions of the edge lengths. A notable exception is when the lengths are distances bewteen random points in Euclidean space, for which the algorithm takes average cubic time.

Keywords: Applied Probability; Combinatorial Optimization; Max Weight Triangle; 3-OPT TSP Neighborhood; Probabilistic Analysis of Algorithms.

1 Introduction

The problem addressed in this paper, called MaxTR, can be stated as follows: given a complete graph of n vertices weighted on the edges, find a triangle of maximum weight. This seemingly innocuous problem, for which there is an obvious polynomial algorithm of complexity $\Theta(n^3)$, does in fact appear, albeit in disguise, as the core question to solve in many optimization problems on graphs but not only. For instance, it relates to finding all-pairs shortest paths, or the best 3-OPT move in TSP local search, but also checking wether a given matrix defines a metric or computing Boolean matrix multiplication over

*Dipartimento di Matematica, Informatica e Fisica, University of Udine, 33100, Udine, Italy. Email: giuseppe.lancia@uniud.it

†Dipartimento di Scienze Economiche e Statistiche, University of Udine, 33100, Udine, Italy. Email: paolo.vidoni@uniud.it

the (OR,AND)-semiring. Therefore, any improvement over the trivial cubic algorithm which enumerates all triangles would have a big impact on many other problems as well. Unfortunately, theory tells us that, if a popular recent conjecture is valid, there cannot be a better-than-cubic algorithm for MaxTR. This result holds also for a set of other classical graph problems that can be reduced to MaxTR. In order to introduce the conjecture, let us start with some definitions.

We say that an algorithm for n-vertex graphs is *truly subcubic* if its runtime is bounded by $O(n^{3-\epsilon})$ for some constant $\epsilon > 0$. Vassilevska-Williams and Williams [12] introduced a framework for relating the truly subcubic solvability of several classic problems to each other. In particular, they defined the notions of *subcubic reducibility* and *subcubic equivalence*. Essentially, a subcubic reduction of a problem P to a problem P' implies that a truly subcubic algorithm for P' becomes a truly subcubic algorithm for P. The problems are subcubic equivalent if each of them can be subcubic reduced to the other. A list of problems that are subcubic equivalent to MaxTR includes, among others ([12, 3]):

- *All pairs Shortest Paths* (APSP): finding a shortest path between each pair of vertices in a weighted graph.
- Finding a triangle of negative weight in a weighted graph.
- Finding a 3-OPT move (replacing 3 edges with 3 new ones) which shortens a TSP tour in a weigthed graph.
- Finding a minimum-weight cycle in a graph of non-negative edge weights.
- Finding the 2nd shortest simple path between two nodes in a weighted digraph.

A recent conjecture, known as the *APSP conjecture* [1], states that there cannot be a truly subcubic algorithm for APSP. The conjecture is widely accepted since a truly subcubic algorithm for APSP would imply, via the reductions described in [12], truly subcubic algorithms for all the above problems, which no one had ever been able to find despite many attempts.

The above list of problems includes 3-OPT, which arises in the context of solving the Traveling Salesman Problem (TSP) via local search heuristics [6, 10]. The 3-OPT input is a TSP tour $T = (\pi_1, \ldots, \pi_n)$. A *move* consists in removing three edges $e_1 := \{\pi_i, \pi_{i+1}\}$, $e_2 := \{\pi_j, \pi_{j+1}\}$, and $e_3 := \{\pi_k, \pi_{k+1}\}$ and replacing them with three new edges f_1, f_2 and f_3 in such a way that $T' := T \setminus \{e_1, e_2, e_3\} \cup \{f_1, f_2, f_3\}$ is still a tour. The move is *improving* if T' is shorter than T. The 3-OPT problem consists in finding any improving move (or the best improving move). Modulo the APSP conjecture, there is no truly subcubic algorithm for 3-OPT, so that the obvious $\Theta(n^3)$ enumeration which tries all possibilities for i, j, k is, in a sense, the best way to find an improving move [3].

The cubic time needed to find an improving 3-OPT move made the usage of the 3-OPT neighborhood for the TSP practically impossible except for very

small graphs. Recently, however, a new technique was proposed in [9], observed to be orders of magnitude faster than the $\Theta(n^3)$ enumeration on all instances on which it was applied, both random and from the standard repository TSPLIB [11]. Let us call $\mathcal{A}_h$ the technique introduced in [9]. The technique is based on a clever enumeration of the moves which tends to find very good moves early on, coupled with a pruning criterion which allows to discard several "bad" moves without having to enumerate them. Given the relation between 3-OPT and MAXTR, we suspected that $\mathcal{A}_h$ could have a similarly effective counterpart for the solution of MAXTR. The focus of this paper has therefore been the study of a version of $\mathcal{A}_h$ adapted for the solution of MAXTR. While a theoretical analysis of the average running time of $\mathcal{A}_h$ for 3-OPT turned out to be too complex (the work [9] contains only experimental evidence but no formal proof of the subcubic running time), in the case of MAXTR we have been able not only to prove that $\mathcal{A}_h$ is truly subcubic on average, but that indeed it is quadratic which is clearly the best possible complexity for an algorithm whose input has size $\Theta(n^2)$.

Since theory told us that any exact algorithm for finding the largest triangle should be at least cubic in the worst case, we looked for (and found) classes of instances on which the algorithm $\mathcal{A}_h$ takes indeed time $\Theta(n^3)$. We also characterized some instances on which it takes (worst-case) $\Theta(n^2)$ time. Finally, we have defined a new randomized algorithm $\mathcal{A}_r$ which is much simpler to implement than $\mathcal{A}_h$ and which also exhibits a truly subcubic average behavior, worse than $\mathcal{A}_h$ but only by a constant factor.

The remainder of the paper is organized as follows. In Section 2 we introduce some useful notation and terminology. Section 3 describes the ideas behind the algorithm $\mathcal{A}_h$ while in Section 4 we describe some best and worst possible instances. Section 5 contains the probabilistic analysis of $\mathcal{A}_h$ as well as some experimental evidence of its quadratic behavior. In Section 6 we describe the randomized algorithm $\mathcal{A}_r$ and compare it to $\mathcal{A}_h$. Section 7 is devoted to the experimental analysis of $\mathcal{A}_h$ on various families of random instances, with both independent and dependent random edge lengths. Some conclusions are drawn in Section 8. Finally we review some useful probabilistic results from the literature in Appendix A.

2 Basic notation and definitions

We consider the complete graph K_n of nodes $\{1, \ldots, n\}$, weighted on the edges. The edges will be denoted preferably by ij, but sometimes also by $\{i, j\}$. Let $\mathcal{T}$ be the set of triples $(i, j, k) \in \{1, \ldots, n\}^3$ such that $i < j < k$. We can think of $\mathcal{T}$ as the set of (vertices of) all triangles in K_n. Clearly $|\mathcal{T}| = \binom{n}{3}$.

Assume the $\binom{n}{2}$ edges of K_n are given costs drawn uniformly at random in $[0, 1]$. In particular for each $1 \leq i < j \leq n$ we have a uniform random variable L_{ij} in $[0, 1]$ representing the length of the edge ij. For each triple $(i, j, k) \in \mathcal{T}$ we define

$$\Delta(i, j, k) := L_{ij} + L_{jk} + L_{ik}.$$

The problem MaxTR studied in this paper consists in finding a triple $(i^*, j^*, k^*) \in \mathcal{T}$ such that

$$\Delta(i^*, j^*, k^*) = \max_{(i,j,k)\in\mathcal{T}} \Delta(i, j, k).$$

We will call any such triple the *best overall*. For $\alpha \in [0, 1]$ let us call α-*good* any edge ij such that $L_{ij} > \alpha$. We extend the definition of α-good to any triple that contains at least one α-good edge. We say that an edge (respectively, a triple) is α-*bad* if it is not α-good.

3 The main idea and the algorithm $\mathcal{A}_h$

There is an obvious $\Theta(n^3)$ algorithm of complete enumeration of all triples, i.e., three nested `for` cycles, for i, j and k, iterating over all triples $(i, j, k) \in \mathcal{T}$, evaluating the Δ-value of each of them and eventually returning the best overall.

In this paper we are going to follow a different strategy, that allows us not to enumerate all triples, but only those who are "good candidates" to be the best overall. The idea is quite simple, and is applied to a sequence of iterative improvements where, at each iteration, there is a certain triple (the current "champion") which is the best we have seen so far and which we want to beat.

Assume the current champion is $\hat{T} = (\bar{i}, \bar{j}, \bar{k})$. Then, for any triple (i, j, k) better than $\hat{T}$ it must be $L_{ij} + L_{jk} + L_{ik} > \Delta(\hat{T})$ and hence

$$\left(L_{ij} > \frac{\Delta(\hat{T})}{3}\right) \vee \left(L_{jk} > \frac{\Delta(\hat{T})}{3}\right) \vee \left(L_{ik} > \frac{\Delta(\hat{T})}{3}\right)$$

i.e., at least one of its edges must be $(\Delta(\hat{T})/3)$-good. Based on this observation, we will set-up an enumeration scheme which builds the triples starting from edges that are $(\Delta(\hat{T})/3)$-good and then completing any such edge into a triple by adding the missing vertex.

Our basic steps are the *selection* and the *expansion* of the edges. The selection of an edge is simply the choice of an edge ij (which has not been selected before). The expansion of ij is the evaluation of all triples $\{i, j, k\}$, for $k \neq i, j$.

Our algorithm, at a high level, can be seen as a sequence of iterations, where each iteration is a selection followed, perhaps, by an expansion. If during an expansion the current champion gets improved, we say that the iteration was *fruitful*, otherwise it was *fruitless*. We will show that, as long as we expand only edges that are $(\Delta(\hat{T})/3)$-good we expect to have, overall, only $O(n)$ expansions. Our first strategy for edge selection is to select them in decreasing order of length. The best data structure for performing this type of selection is a heap (justifying the "h" in $\mathcal{A}_h$), from which we pop out the edges from the largest to the smallest.

Since in our algorithms we constantly make references to the threshold $(\Delta(\hat{T})/3)$ defined by the current champion $\hat{T}$, from now on we will say that an edge (or a triple) is *good* if it is $(\Delta(\hat{T})/3)$-good, and it is *bad* otherwise.

Procedure 1 DeterministicHeapBasedAlgorithm $\mathcal{A}_h$

1. Build a max-heap with elements $[(i,j), L_{ij}]$ $\forall 1 \leq i < j \leq n$ sorted by L-values;
2. Set $\hat{T} := \emptyset$ and $\Delta(\hat{T}) := -\infty$;
3. **while** the L-value of the top heap element is $> \Delta(\hat{T})/3$ **do**
4. Extract the top of heap, let it be $[(a,b), L_{ab}]$;
5. let $c = \text{argmax}_{k \neq a,b}(L_{ab} + L_{bk} + L_{ak})$;
6. **if** $L_{ab} + L_{bc} + L_{ac} > \Delta(\hat{T})$ **then**
7. $\hat{T} := \{a, b, c\}$; /* update the champion */
8. **endif**
9. **endwhile**
10. **return** T^*;

The deterministic algorithm $\mathcal{A}_h$ is described in Procedure 1. In this algorithm we make use of a max-heap, in which we put all the edges with their L-value. Each entry of the heap is indeed a 2-field record $[(a,b), L_{ab}]$. The max heap is basically a binary tree whose nodes correspond to the pairs ij, and such that, for each node ab, it is $L_{ab} \geq L_{xy}$ for all xy in the subtree rooted at ab. Building the heap (step 1) is done in linear time w.r.t. the number of heap elements (i.e., in time $\Theta(n^2)$ in our case) by using the standard procedure `heapify`() (described in each classic book on algorithms and data structures such as, e.g., [4]).

At the beginning $\hat{T}$ is undefined and we set $\Delta(\hat{T}) := -\infty$ as the value to beat. Testing if there are still any good edges is done in step 3 and takes time $O(1)$ per test since we just need to read the root of the heap. The selection is done in step 4 and takes time $O(\log n^2) = O(\log n)$ to maintain the heap property. The main loop 3–8 terminates as soon as there are no longer any good edges.

At the generic step, we pop the top of the heap, let it be $[(a,b), L_{ab}]$. If $L_{ab} \leq \Delta(\hat{T})/3$, we stop and return $\hat{T}$ as the best triple possible. Otherwise, a and b are two out of the three indices of some potential triple better than $\hat{T}$. Knowing two out of the three indices, we then run the expansion (step 5) which, in linear time, finds the best completion of $\{a, b\}$ into a triple. Each time we find a triple better than the current champion, we update $\hat{T}$. This way the termination condition becomes easier to satisfy and we get closer to the end of the loop.

If, overall, there are N selections (and, therefore N expansions), the running time of the algorithm is $O(n^2 + N(\log n + n))$ which is $O(n^2)$ as long as $N = O(n)$.

4 Best- and worst-case analysis

Any algorithm for finding a best triangle in a graph must have a complexity $\Omega(n^2)$, since, at the very least, all edges have to be considered for being in the best triangle. In this section we show that our algorithm has a best-case complexity of $O(n^2)$ and a worst-case complexity of $\Theta(n^3)$.

Theorem 1 *The algorithm $\mathcal{A}_h$ has a best-case complexity $O(n^2)$.*

Proof: The simplest best-case instance is when all the edge lenghts are the same, say $= 1$. In this graph, any triangle is a best triangle and has value 3. The algorithm starts by selecting an edge and finding, in the expansion, as the first champion the best overall. This will set the threshold for good edges at $3/3 = 1$ so that there will be no longer any good edge. At this point the algorithm stops. The total time is $O(n^2)$ for building the heap plus $O(\log n + n)$ for the single loop iteration. □

It is easy to show that there are also best-case instances in which all costs are distinct. For example, set $L_{12} = 1$ and take all the remaining lenghts in $[0, \frac{1}{3}]$. Also here, the best triangle is found at the first, and only, expansion.

Let us now look at a worst-case example.

Theorem 2 *The algorithm $\mathcal{A}_h$ has a worst-case complexity $\Theta(n^3)$.*

Proof: Clearly the complexity is $O(n^3)$ since there are $O(n^2)$ elements in the heap and the expansion of each of them takes $O(n)$ time. To show the lower bound $\Omega(n^3)$ consider the following instance.

Fix any $\epsilon \in (0, 1/2)$ and define

$$L_{ij} = \begin{cases} 1 - \epsilon & \text{if } i = j \pmod 2 \\ 1 & \text{if } i \neq j \pmod 2 \end{cases}$$

Notice that each triangle contains at least one pair of nodes of the same parity, and hence at least one edge of cost $1 - \epsilon$. Therefore, the optimal value is $3 - \epsilon$ and is realized by any triple in which exactly two of the nodes have the same parity.

The algorithm selects any edge of value 1 at the first iteration (e.g. $\{1, 3\}$), and, at the first expansion, finds the best overall (e.g., $(1, 2, 3)$). This sets the threshold for being a good edge at $1 - (1/3)\epsilon$ so that every edge between an even and an odd vertex is good. Notice that there are $n^2/4 = \Omega(n^2)$ such edges. All these edges will be selected and expanded, in fruitless iterations, for a total work of $\Omega(n^3)$. □

Considering that the worst-case complexity is $\Theta(n^3)$, one might wonder if the hidden multiplicative constant can be bigger than 1. Indeed, one could say that since a triple contains three edges, it could be evaluated three times (if each of these edges is good). However, we show that it will be evaluated at

most twice. Furthermore, only triples that are good (at a given point of the algorithm) can be evaluated twice, not all triples.

Lemma 1 *Let t be the number of good triples at any point during the execution of the algorithm $\mathcal{A}_h$. Then the number of triples evaluated by the algorithm from this point on is bounded by $2t$.*

Proof: We know that only (a subset of) the t good triples will be evaluated from now on. However a specific triple $\{i, j, k\}$ could be evaluated more than once, since it can be good because of ij OR jk OR ik. The only good triples that could be evaluated three times are those whose edges are all good. Let us consider any such good triple $T = \{i, j, k\}$, i.e.,

$$(L_{ij} > \Delta(\hat{T})/3) \wedge (L_{jk} > \Delta(\hat{T})/3) \wedge (L_{ik} > \Delta(\hat{T})/3).$$

Without loss of generality, assume $L_{ij} \geq L_{jk} \geq L_{ik}$. Whenever ij is popped from the heap and T gets evaluated for the first time, the value $\Delta(T) = L_{ij} + L_{jk} + L_{ik} > \Delta(\hat{T})$ and so the champion gets updated to $\hat{T} := T$ and its value to $\Delta(\hat{T}) := L_{ij} + L_{jk} + L_{ik}$. Note that at this point

$$L_{ik} = \min\{L_{ij}, L_{jk}, L_{ik}\} \leq \frac{L_{ij} + L_{jk} + L_{ik}}{3} = \frac{\Delta(\hat{T})}{3}$$

and hence the element ik will not be popped from the heap for expansion, since the search will stop earlier. Therefore, T cannot be evaluated three times. □

5 The average-case complexity of the algorithm: probabilistic findings and empirical evidence

Each run of the algorithm on a random instance determines some random variables, such as:

- the total number N of expansions, which corresponds to the number of selections and loop iterations;
- the L-values $H_1 > H_2 > \cdots > H_N$ of the 1st, 2nd, ..., last edge pulled from the heap (since the random variables are continuous, they are strictly decreasing almost surely);
- the total number C of times that the champion $\hat{T}$ (and its value) gets updated;
- the values $V_1 < V_2 < \cdots < V_C$ of the champion after the 1st, 2nd, ..., last update;
- the total number M of triples evaluated by the algorithm and defined as $M = N(n-2)$ (this is the quantity which determines the complexity of the algorithm).

The aim here is to determine the expected total number of triples $E(M) = E(N)(n-2)$, as a function of n, and then to discuss the average-case complexity of the algorithm. By considering a suitable mean pattern, we obtain an upper bound for $E(M)$ and we find that, as it appears experimentally, $E(M) = O(n^2)$. Hereafter, we define as a step of the algorithm the total number of selections and loop iterations which produces a new champion, and the number C of times that the champion gets updated corresponds to the number of steps required to terminate the algorithm. Furthermore, we usually consider the interest quantities up to terms of order $O(n^{-1/2})$.

5.1 The steps of the deterministic algorithm

Step 1. In the initial step of the algorithm, since $\Delta(\hat{T}) := -\infty$, the (relative) frequency of good edges in the heap is $p_1 = 1$ and then the number of available edges, which can specify the first champion, is simply $R_1 = n(n-1)/2$. The value of the first edge pulled from the heap is $H_1 = \max_{1 \le i < j \le n}\{L_{ij}\}$, namely, the maximum of $m = n(n-1)/2$ independent uniform random variables in $[0,1]$ L_{ij}, $i, j = 1, \ldots, n$, $i < j$, and then the value of the first champion is given by

$$V_1 = H_1 + B_1.$$

Here we set $B_1 = B|(L_{wv} < E(H_1), w = a, b)$, where

$$B := \max_{v=1,\ldots,n; v \neq a,b}\{L_{av} + L_{bv}\}$$

is the maximum of $m = n-2$ independent triangular random variables, obtained as the sum of pairs of independent uniform random variables in $[0,1]$. Since H_1 corresponds to the maximum edge length, say realized by the pair (a,b), the random variable B has to be considered under the condition that the random variables L_{wv}, $w = a, b$, assume a value that is lower than the observed value of L_{ab}.

From Theorem A2, we state that

$$E(H_1) = \frac{m}{m+1} = 1 - \frac{1}{1 + n(n-1)/2} = 1 - \frac{2}{n^2 - n + 2} = 1 + O(n^{-2})$$

and, applying relation (A3), we conclude that $E(B_1) = E(H_1)\,E(B)$. Since from (A2) a lower bound for $E(B)$ can be specified, a simple lower bound for the expected value of the first champion is given by

$$\begin{aligned} E(V_1)^- &= \left(1 - \frac{2}{n^2 - n + 2}\right)\left\{3 - \frac{2^{-(n-2)}}{2n-3} - \sqrt{\frac{\pi}{2(n-2)}}\right\} \\ &= 3 - \sqrt{\frac{\pi}{2}}\, n^{-1/2} + o(n^{-1/2}). \end{aligned}$$

Step *r*. We consider the generic r-th step, with $r = 2, 3, \ldots$, and the interest quantities are specified as a function of those ones obtained in the preceding

step, with the aim of defining a suitable iterative procedure. Let us assume that the lower bound for the expected value of the champion defined in the $(r-1)$-th step is

$$E(V_{r-1})^- = 3 - \lambda_{r-1}\sqrt{\frac{\pi}{2}}\, n^{-1/2} + o(n^{-1/2}), \quad r = 2, 3, \ldots,$$

with $\lambda_1 = 1$ and $\lambda_r = g(\lambda_{r-1})$, $r = 2, 3, \ldots$, where $g(\cdot)$ is a function to be defined in the following. Then, the (relative) frequency of good edges in the heap is approximated by

$$p_r = P\left(L_{ij} > \frac{E(V_{r-1})^-}{3}\right) = 1 - \frac{E(V_{r-1})^-}{3} = \frac{\lambda_{r-1}}{3}\sqrt{\frac{\pi}{2}}\, n^{-1/2} + o(n^{-1/2}),$$

and an upper bound for the expected number of good edges in the heap is

$$R_r = \frac{n(n-1)}{2}\, p_r = \frac{n^{1/2}(n-1)}{6}\, \lambda_{r-1}\sqrt{\frac{\pi}{2}} + o(n^{3/2}), \quad r = 2, 3, \ldots . \tag{1}$$

If there is at least one good edge available in the heap, it could specify a new champion with value

$$V_r = H_r + B_r,$$

where $H_r = X_{(m-E_{r-1})}$, with $m = n(n-1)/2$, is the $(m - E_{r-1})$-th record of the heap in decreasing order. Here,

$$E_{r-1} = \sum_{s=1}^{r-1} E(N_s) \tag{2}$$

is the expected number of expansions performed until the $(r-1)$-th step, where N_s is the number of expansions produced in the s-th step. Clearly, $N_1 = 1$, whereas the random variables N_s, $s = 2, 3, \ldots$, will be defined in the following. Whenever $E_{r-1} = o(n^{3/2})$, as usual in the observed behavior of the algorithm, we may conclude, using Theorem A2, that

$$E(H_r) = \frac{m - E_{r-1}}{m+1} = 1 - \frac{E_{r-1}+1}{1 + n(n-1)/2} = 1 + o(n^{-1/2}).$$

Moreover, $B_r = B|B > \beta_r$, where B is the maximum of $m = n-2$ independent triangular random variables and

$$\beta_r = E(V_{r-1})^- - E(H_r) = 2 - \lambda_{r-1}\sqrt{\frac{\pi}{2}}\, n^{-1/2}$$

is the value for B which has to be exceeded in order to beat the current champion. Applying relation (A5), with $\beta = \beta_r$, we find out that the probability of this event is

$$P(B > \beta_r) = 1 - \left(1 - \lambda_{r-1}^2 \frac{\pi}{4}\, n^{-1}\right)^{n-2}.$$

It is immediate to conclude that the number of expansions required in the r-th step in order to obtain a new champion is a random variable N_r following a geometric distribution with expected value

$$E(N_r) = \frac{1}{P(B > \beta_r)}, \quad r = 2, 3, \ldots. \tag{3}$$

If the (upper bound of the) expected number of good edges in the heap is strictly lower that expected number of expansions to be performed until the r-th step, that is, if

$$R_r - E_r < 0, \tag{4}$$

then the algorithm terminates. In this case we have a number of available edges in the heap $R_r - \sum_{s=1}^{r-1} E(N_s)$ which is strictly lower than the expected number of expansions $E(N_r)$ to be performed in r-th step in order to get a new champion. Then, the algorithm performs $r-1$ steps (which is an approximation for the number C of times that the champion gets updated), the (lower bound for the) expected value of the champion is $E(V_{r-1})^-$ and an upper bound for the mean value of the total number of expansions $E(N)$ is

$$\max\left\{\sum_{s=1}^{r-1} E(N_s), R_r\right\}.$$

We note in passing that this quantity is not greater than R_{r-1}, namely, the expected number of good edges in the heap at the beginning of the $(r-1)$-th step.

If the termination condition (4) is not satisfied, we expect that the champion and its value get updated in the r-th step. Using (A4), we compute the following lower bound for the expectation of B_r

$$E(B_r)^- = E(B|B > \beta_r)^- = 2 - g(\lambda_{r-1})\sqrt{\frac{\pi}{2}}\, n^{-1/2} + o(n^{-1/2})$$

and, consequently, a lower bound for the expected value of the potential champion defined in the r-th step

$$E(V_r)^- = E(H_r) + E(B_r)^- = 3 - g(\lambda_{r-1})\sqrt{\frac{\pi}{2}}\, n^{-1/2} + o(n^{-1/2}),$$

where, up to terms of order $O(n^{-1/2})$,

$$g(\lambda) = \frac{1}{1 - \{1 - \lambda^2(\pi/4)n^{-1}\}^{n-2}}\left[2\Phi\left(\lambda\sqrt{\frac{\pi}{2}}\right) - 1 - \lambda\{1 - \lambda^2(\pi/4)n^{-1}\}^{n-2}\right],$$

with $\lambda \in (0, 1]$ and $\Phi(\cdot)$ the distribution function of a standard normal random variable. Since

$$\{1 - \lambda^2(\pi/4)n^{-1}\}^{n-2} \geq e^{-\lambda^2\pi/4},$$

it is easy to state that function

$$\tilde{g}(\lambda) = \frac{1}{1 - e^{-\lambda^2 \pi/4}} \left\{ 2\Phi\left(\lambda\sqrt{\frac{\pi}{2}}\right) - 1 - \lambda e^{-\lambda^2 \pi/4} \right\}$$

is an upper bound for $g(\lambda)$ and also a good approximation, for moderate values on n. Notice that function $g(\cdot)$ is increasing on the domain $(0, 1]$ and its right limit at $\lambda = 0$ is 0. Moreover, the sequence λ_r, $r = 1, 2, \ldots$, with initial value $\lambda_1 = 1$, is decreasing towards 0 and the associated sequence $R_r - E_r$, $r = 1, 2, \ldots$, usually assumes negative values after a few number of steps.

5.2 The average-case complexity of $\mathcal{A}_h$

In this section, we discuss the asymptotic order, as the number n of nodes increases, of a suitable upper bound for the expected number of times $E(C)$ that the champion gets updated and for the expected number of expansions $E(N)$ produced in the development of the algorithm. These results enable an approximate evaluation of the average-case complexity of the algorithm, by assuming that its mean behavior is the one outlined in the previous section.

Theorem 3 *Let K_n be a complete graph with $n \geq 3$ nodes, where the $n(n-1)/2$ edge weights are independent uniform random variables in $[0, 1]$. By considering the mean behavior of the algorithm outlined in Section 5.1, if*

$$r^\star = \max\{r \in \mathbf{N}^+ \,:\, R_r - E_r \geq 0\},$$

with R_r and E_r specified in (1) and (2), respectively, we have that $r^\star = O(\log n)$ and $R_{r^\star} = O(n)$.

Proof: At first, from equation (3), we state immediately that $E(N_r) \geq 1$, $r = 1, 2, \ldots$, and then, since $R_r - E_r \leq R_r - r$, we may conclude that $r^\star \leq \hat{r}$, with $\hat{r} \in \mathbf{N}^+$ such that $R_{\hat{r}} - \hat{r} = 0$. Here, for simplifying the exposition, we assume that the equality turns out to be exactly true. Using (1), and neglecting terms of lower asymptotic order, this last equation can be rewritten as

$$\frac{\hat{r}}{\lambda_{\hat{r}-1}} = \frac{n^{1/2}(n-1)}{6}\sqrt{\frac{\pi}{2}}. \tag{5}$$

Moreover, by considering the definition of function $g(\cdot)$, introduced at the end of Section 5.1, since $2\Phi(\lambda\sqrt{\pi/2}) < 1+\lambda$, for $\lambda \in (0, 1]$, it is easy to show that there exists a quantity $b \in (0, 1)$ such that $g(\lambda) \leq b\lambda$, for $\lambda \in (0, 1]$. Recalling that the sequence λ_r, $r = 1, 2, \ldots$, is obtained by applying recursively function $g(\cdot)$, starting from $\lambda_1 = 1$, we state that $\lambda_{\hat{r}-1} \leq b^{\hat{r}-2}$. Thus, the following inequalities hold

$$\frac{1}{b^{\hat{r}-2}} \leq \frac{\hat{r}}{b^{\hat{r}-2}} \leq \frac{\hat{r}}{\lambda_{\hat{r}-1}} = \frac{n^{1/2}(n-1)}{6}\sqrt{\frac{\pi}{2}},$$

and applying the logarithmic transformation to the first and the last terms, we conclude that $\hat{r} = O(\log n)$. Finally, since $r^\star \leq \hat{r}$, we prove also that $r^\star = O(\log n)$.

The asymptotic order of $R_{r^\star}$ may be easily derived by considering that $r^\star \leq n$ and that, from (5), we obtain the following upper bound

$$\lambda_{\hat{r}-1} \leq \frac{6n}{n^{1/2}(n-1)\sqrt{\pi/2}}.$$

Plugging this inequality in equation (1), we also prove that $R_{r^\star} = O(n)$. □

This theorem sheds some light on the average-case complexity of the deterministic algorithm $\mathcal{A}_h$, since $r^\star$ can be viewed as an approximation for $E(C)$, namely, the expected number of times that the champion gets updated, and $R_{r^\star}$ is an upper bound for the overall expected number of expansions $E(N)$. From Theorem 3, we conclude that $E(M) = O(n^2)$ and that $E(C) = O(\log n)$, namely the average-case complexity of the algorithm turns out to be $\Theta(n^2)$ and the expected number of times that the champion gets updated is $\Theta(\log n)$.

These theoretical findings are in accordance with the results of the following simulation study. We consider a set of runs of the algorithm on random instances of size varying from $n = 20$ to $n = 5120$, each time doubling the value of n. The values reported in each row of Table 1 are the averages over T_n instances of size n, where $T_n = 10,000$ for $n \leq 500$ and $T_n = 2,000$ for $n > 500$.

n	$\hat{E}(N)$	ρ_N	$\hat{E}(C)$	δ_C	$\hat{E}(M)$	$\binom{n}{3}$	ratio
20	10.18	-	2.32	-	183	1,140	6,2
40	20.07	1,97	2.96	0.64	763	9,880	12,9
80	39.99	1.99	3.61	0.65	3,119	82,160	26.3
160	79.10	1.98	4.30	0.69	12,498	669,920	53.6
320	158.01	2.00	4.97	0.67	50,247	5,410,240	107.6
640	317.80	2.01	5.66	0.69	202,759	43,486,080	214.4
1280	629.64	1.98	6.39	0.73	804,674	348,706,560	433.3
2560	1,259.85	2.00	7.02	0.63	3,222,692	2,792,926,720	866.6
5120	2,486.33	1,97	7.72	0.70	12,725,050	22,356,515,840	1756.8

Table 1: Empirical mean values, computed using T_n simulated instances of size n, of the random variables involved in the development of the algorithm $\mathcal{A}_h$.

Column $\hat{E}(N)$ is the estimated mean value of loop iterations, i.e., of elements pulled from the heap. The estimated total number of triples evaluated (rounded to integer), is shown in column $\hat{E}(M)$. Next to this column we report the value of $\binom{n}{3}$ which is the number of triples evaluated by the complete enumeration algorithm. We can appreciate how our method achieves speed-ups from 1 up to 3 orders of magnitude. In column ρ_N we report the ratio of $\hat{E}(N)$ for consecutive-size instances and we see how this value is approximately 2, i.e., $\hat{E}(N)$ appears to be linear in n. Consequently, $\hat{E}(M) = \hat{E}(N)(n-2)$ appears to be quadratic

and then we can say that, on average, the algorithm evaluates more or less one triple for each pair of nodes.

Column $\hat{E}(C)$ is the estimated total number of times that the champion has been updated. Each time we double n, $\hat{E}(C)$ increases by roughly a constant factor. In column δ_C we report the difference of $\hat{E}(C)$ for instances of consecutive size. Similarly to ρ_N, also this column seems to revolve around a constant, roughly $\simeq 0.69$. This growth is typical of a logarithmic function. Hence we may conclude that these empirical findings are in accordance to the theoretical statements assuring that $E(M) = O(n^2)$ and $E(C) = O(\log n)$.

Finally, in Table 2 we report the values for $E(N)$ and $E(C)$ obtained by considering the associated approximations $R_{r^\star}$ and $r^\star$ outlined in Section 5.1. The number of nodes varies, as before, from $n = 20$ to $n = 5120$. Since these approximations correspond to integer values, in order to obtain results comparable to those ones given in Table 1, we specify $E(N)$ as a suitable weighted mean between $R_{r^\star}$ and $R_{r^\star+1}$. Moreover, we compute also $E(M) = E(N)(n-2)$. The values reported in Table 2 are similar to those one obtained using simulations and they exhibit the same behavior as n increases.

n	$E(N)$	$E(C)$	$E(M)$
20	9.50	3	171
40	18.88	4	717
80	37.28	5	2,908
160	78.89	5	12,465
320	166.88	6	53,068
640	340.53	7	217,258
1280	669.03	8	855,020
2560	1,258.96	9	3,220,420
5120	2,442.17	10	12,499,026

Table 2: Approximations for $E(N)$, $E(C)$ and $E(M)$, for different values of n, obtained using the results presented in Section 5.1.

6 A randomized algorithm for MaxTR

The deterministic algorithm $\mathcal{A}_h$ which performs so well on average is based on two main ideas: (i) selecting the edges in order of length (the rationale being that to obtain a "large" triangle at least one edge should be "large"), with the hope of obtaining good champions early on; (ii) employing a pruning criterion which does not consider some edges for expansion (i.e., does not generate some triples which cannot beat the champion). A question then arises about which between (i) and (ii) is the winning idea –if any– or if they are both needed.

The answer is that the winning idea is (ii), while (i) is important only to speed-up the algorithm by a constant factor, but not to change its order of complexity. We have come to this conclusion by investigating a new, simpler,

algorithm which is the randomized version of $\mathcal{A}_h$ and which we called $\mathcal{A}_r$. The algorithm $\mathcal{A}_r$ is outlined in Procedure 2. In this algorithm we do not employ a heap to select the edges in order of length, but we simply select them at random. The main loop consists of $\binom{n}{2}$ iterations. At each iteration the selection of an edge (step 4) has cost $O(1)$ (considering the generation of a random number an $O(1)$ operation). The expansion is done in step 7. Assuming there are N expansion altogether, the running time of this algorithm is $O(n^2 + Nn)$, which is $O(n^2)$ as long as $N = O(n)$.

Procedure 2 RANDOMIZEDALGORITHM $\mathcal{A}_r$

1. Set $\mathcal{P} := \{\{i, j\} : 1 \leq i < j \leq n\}$;
2. Set $\hat{T} := \emptyset, \Delta(\hat{T}) := -\infty$; /* Starting undefined champion */
3. **while** $\mathcal{P} \neq \emptyset$ **do**
4. Pick $p = \{a, b\} \in \mathcal{P}$ at random;
5. $\mathcal{P} := \mathcal{P} \setminus \{p\}$;
6. **if** $L_{ab} > \Delta(\hat{T})/3$ **then**
7. let $c = \text{argmax}_{k \neq a,b}(L_{ab} + L_{bk} + L_{ak})$;
8. **if** $L_{ab} + L_{bc} + L_{ac} > \Delta(\hat{T})$ **then**
9. $\hat{T} := \{a, b, c\}$; /* update the champion */
10. **endif**
11. **endif**
12. **endwhile**
13. **return** $\hat{T}$;

Since our analysis of the average running time of $\mathcal{A}_h$ has determined that $\mathcal{A}_h$ is optimal for the problem, we did not consider necessary to perform a similar evaluation for $\mathcal{A}_r$, but rather to assess experimentally its performance and compare it to that of $\mathcal{A}_h$. In Table 3 we report the same type of data as in Table 1 but this time for the algorithm $\mathcal{A}_r$. Also in this case we notice that $\hat{E}(N)$ is linear, $\hat{E}(M)$ is quadratic and $\hat{E}(C)$ is logarithmic.

Finally, if we compare the empirical results for $\mathcal{A}_h$ and $\mathcal{A}_r$, by computing the ratio between the number of triples evaluated by the latter and the former, we notice that the $\mathcal{A}_r$ appears to be slower than $\mathcal{A}_h$ by approximately a constant factor around 25% (see Table 4).

7 Experiments with various types of instances

The goal of this section is to show (empirically) that our algorithm performs well not only on instances with uniform random costs but also on instances where the costs follow some other distributions. In particular, in a first experiment we have considered some classical probability distributions, such as Normal, Log-normal, Beta, etc, and have generated the edge lengths as independent random

n	$\hat{E}(N)$	ρ_N	$\hat{E}(C)$	δ_C	$\hat{E}(M)$	$\binom{n}{3}$	ratio
20	13.77	-	3.99	-	248	1,140	4.5
40	26.41	1.92	5.04	1.05	1,004	9,880	9.8
80	51.34	1.94	6.07	1.03	4,004	82,160	20.5
160	100.39	1.96	7.12	1.05	15,862	669,920	42.2
320	199.25	1.98	8.18	1.06	63,363	5,410,240	85.3
640	393.31	1.97	9.28	1.10	250,933	43,486,080	173.2
1280	788.78	2.00	10.29	1.01	1,008,056	348,706,560	345.9
2560	1568.66	1.99	11.36	1.07	4,012,645	2,792,926,720	696.0
5120	3157.03	2.01	12.19	0.83	16,157,687	22,356,515,840	1383.6

Table 3: Empirical mean values, computed using T_n simulated instances of size n, of the random variables involved in the development of the algorithm $\mathcal{A}_r$.

n	$\mathcal{A}_h$	$\mathcal{A}_r$	ratio
20	183	248	1.35
40	763	1,004	1.31
80	3,119	4,004	1.28
160	12,498	15,862	1.27
320	50,247	63,363	1.26
640	202,759	250,933	1.24
1280	804,674	1,008,056	1.25
2560	3,222,692	4,012,645	1.25
5120	12,725,050	16,157,687	1.27

Table 4: Comparing $\hat{E}(M)$ for $\mathcal{A}_h$ and $\mathcal{A}_h$

variables with those distributions. The results, described in Section 7.1 show how the work done by the algorithm grows as a quadratic function of n on all the distributions we tried.

In a second experiment we have studied the behavior of our algorithm on instances where the edge costs are not independent. In particular, we have considered *Euclidean* instances, i.e., instances in which the nodes of the graph are random points of $\mathbb{R}^d$ and the length of the edges are the Euclidean distances between the nodes. The results, described in Section 7.2, show how the work done by the algorithm still grows as a quadratic function of n on all the distributions we tried, with the notable exception of points whose coordinates are uniform random variables in $[0, 1]$. Indeed, we prove that for such uniform Euclidean instances, the algorithm has a cubic complexity on average (although it can be still more effective than complete enumeration by a constant multiplicative factor).

How to read the tables. The results are reported in a set of tables organized as follows. For each value of $n = 20, 40, \ldots, 5120$ we have generated 1000 random instances. To each instance type we associate a column of a table, labeled by the corresponding distribution of the random costs (for instance, the label $N(0,1)$ refers to a normal distribution with mean 0 and standard deviation 1). For each value of n we report the average number of triples (rounded to integer) evaluated by the algorithm. The growth of the values along this column illustrates the average complexity of the algorithm on these instances. The next column, labeled ρ, reports the ratio between the number of triples evaluated for n and for $n/2$. By looking at this column we get an immediate idea about the fact that the growth is cubic, quadratic, or of some other order. Since the value of n doubles each time, for a cubic algorithm the ratio should tend, asymptotically, to 8. For a quadratic algorithm the ratio should tend to 4. A ratio smaller than 4 is also possible. Of course this does not mean that the algorithm takes a less than quadratic time to solve the problem, which would be impossible. The algorithm is still quadratic, but the quadratic part is spent in building the heap, while, once the heap has been built, the time to find the best triple is less than quadratic. If $\rho \simeq 2$ then the best triple is found in linear time, typically by just one expansion. A final row, labeled 'ratio', reports the ratio between the total number of triples and the number of triples evaluated by $\mathcal{A}_h$ for the last value of n. This number is indicative of the speed-up, with respect to the time spent in evaluations, of our method over the complete enumeration approach for $n \simeq 5000$.

7.1 Independent random costs

In this experiment we have considered the following distributions:

$N(\mu, \sigma)$: `Normal`, with mean μ and standard deviation σ. In particular, we have experimented with $(\mu, \sigma) = (0, 1)$.

$LN(\mu, \sigma)$: `Lognormal`, with mean μ and standard deviation σ of the natural logarithm transformation. In particular, we have experimented with $(\mu, \sigma) = (1, 0.1)$ and $(\mu, \sigma) = (1, 2)$.

$t(\nu)$: `Student` t, with ν degrees of freedom. In particular, we have experimented with $\nu = 3$ and $\nu = 5$.

$\beta(a, b)$: `Beta` with parameters a, b. In particular, we have experimented with $(a, b) = (0.5, 0.5)$, $(a, b) = (2, 2)$, $(a, b) = (0.5, 2)$ and $(a, b) = (2, 0.5)$.

Each distribution has its own important parameters, which, clearly, in our runs we had to fix to a limited number of (hopefully representative) cases. For the normal distribution, however, the study of our algorithm with $\mu = 0, \sigma = 1$ catches the behavior for all possible values of μ, σ. This is due to the following result:

Lemma 2 *The execution of $\mathcal{A}_h$ is invariant with respect to an affine transformation of the input costs (i.e., when each cost L_{ij} is replaced by $L'_{ij} := A+BL_{ij}$, for constants $A \in \mathbb{R}$ and $B > 0$).*

Proof: Notice that $L'_{ij} \geq L'_{hk}$ if and only if $L_{ij} \geq L_{hk}$ and hence the edges are popped from the heap in the same order as before. Furthermore, if we define $\Delta'(i,j,k) := L'_{ij} + L'_{jk} + L'_{ik}$ we have $\Delta'(i,j,k) = 3A + B\Delta(i,j,k)$ for each triangle (i,j,k), so that (i,j,k) is a better triangle than (u,v,w) with respect to L' if and only if it was a better triangle also with respect to L. Finally, since

$$\frac{\Delta'(\bar{i},\bar{j},\bar{k})}{3} = \frac{3A + B\Delta(\bar{i},\bar{j},\bar{k})}{3} = A + B\,\frac{\Delta(\bar{i},\bar{j},\bar{k})}{3}$$

for the good edges it is

$$L'_{ij} > \frac{\Delta'(\bar{i},\bar{j},\bar{k})}{3} \iff A + BL_{ij} > A + B\,\frac{\Delta(\bar{i},\bar{j},\bar{k})}{3} \iff L_{ij} > \frac{\Delta(\bar{i},\bar{j},\bar{k})}{3}.$$

Therefore the good edges are the same and the sequence of champions and expansions is the same for L' and L. □

We have the following corollary:

Corollary 1 *For any $\mu \in \mathbb{R}$ and $\sigma > 0$, the average running time of $\mathcal{A}_h$ over instances with costs distributed according to $N(\mu,\sigma)$ is the same as when the costs are distributed according to $N(0,1)$.*

Proof: This follows immediately by noticing that for a random variable X with distribution $N(\mu,\sigma)$ it is $X = \mu + \sigma Y$ where Y is a random variable with distribution $N(0,1)$. □

By the same observation, our analysis of uniform distributions, which was made for uniform random variables Y in $[0,1]$, applies to uniform random variables X in $[a,b]$ for each $a < b$, since in this case $X = a + (b-a)Y$.

The results of the computational experiments, reported in Table 5, show that our algorithm is quadratic over all distributions examined. Given that on all instances we must pay a quadratic time for the construction and usage of the heap, let us just focus on the part concerning the expansions of edges and evaluations of triples. For the normal distribution instances, the number of triples evaluated grows slightly slower than a quadratic function. A similar growth is exhibited by the Lognormal instances with $\mu = 1, \sigma = 0.1$. For Lognormal instances with $\mu = 1, \sigma = 2$ as well as for Student t instances with $\nu = 3$ and $\nu = 5$, the number of triples evaluated is even smaller, and appears to grow almost linearly in n. An explanation of this phenomenon is that there is a not negligible probability of observing both high and low edge costs. The beta distribution instances, for all values of a and b that we tried, are those for which the number of triples evaluated grows almost perfectly as a $\Theta(n^2)$ function. In particular, for $a = b = 2$ whenever n doubles, the number of triples evaluated is almost exactly 4 times the value that it was before.

n	$\binom{n}{3}$	$N(0,1)$	ρ	$LN(1,0.1)$	ρ	$LN(1,2)$	ρ
20	1140	122	-	114	-	45	-
40	9880	463	3.8	416	3.6	119	2.6
80	82160	1713	3.7	1620	3.9	313	2.6
160	669920	6570	3.8	6226	3.8	797	2.5
320	5410240	24155	3.7	22636	3.6	1941	2.4
640	43486080	85968	3.6	84197	3.7	4978	2.6
1280	348706560	305586	3.6	323732	3.8	11471	2.3
2560	2792926720	1005527	3.3	1208520	3.7	26325	2.3
5120	22356515840	3411357	3.4	4686272	3.9	57742	2.2
ratio	-	6554×	-	4771×	-	387179×	-

n	$\binom{n}{3}$	$t(3)$	ρ	$t(5)$	ρ	$\beta(0.5,0.5)$	ρ
20	1140	48	-	64	-	211	-
40	9880	117	2.4	167	2.6	906	4.3
80	82160	265	2.3	424	2.5	3661	4.0
160	669920	574	2.2	1037	2.4	14752	4.0
320	5410240	1316	2.3	2392	2.3	58587	4.0
640	43486080	2677	2.0	5791	2.4	238581	4.1
1280	348706560	5979	2.2	12579	2.2	950841	4.0
2560	2792926720	12571	2.1	28420	2.3	3795768	4.0
5120	22356515840	28677	2.3	51058	1.8	14621374	3.9
ratio	-	779597×	-	437865×	-	1529×	-

n	$\binom{n}{3}$	$\beta(0.5,2)$	ρ	$\beta(2,2)$	ρ	$\beta(2,0.5)$	ρ
20	1140	146	-	154	-	222	-
40	9880	630	4.3	658	4.3	914	4.1
80	82160	2485	3.9	2583	3.9	3659	4.0
160	669920	10342	4.2	10398	4.0	14728	4.0
320	5410240	40315	3.9	41185	4.0	59098	4.0
640	43486080	161418	4.0	165431	4.0	232610	3.9
1280	348706560	650200	4.0	661346	4.0	938734	4.0
2560	2792926720	2632878	4.0	2655056	4.0	3559253	3.8
5120	22356515840	10598903	4.0	10609394	4.0	12028831	3.4
ratio	-	2109×	-	2107×	-	1859×	-

Table 5: Statistics for various distributions of the random costs. Averages over 1000 instances for each value of n.

7.2 Random points in the Euclidean space

In this experiment we consider graphs embedded in the Euclidean space $\mathbb{R}^d$ (the experiments reported in this section refer mostly to the dimension $d = 2$, i.e., the graph lies in the plane. The choice of $d = 2$ is representative of the situation for each fixed dimension). Each instance, which we call a *Euclidean instance*, is

created by generating n points $x^1, \ldots, x^n \in \mathbb{R}^d$, where each coordinate x_i^j, for $i = 1, \ldots, d$, $j = 1, \ldots, n$, is drawn, independently from the others, according to a fixed distribution D (such as uniform, Normal, etc.). We then take as lengths of the edges $L_{ij} = ||x^i - x^j||$. Clearly for Euclidean instances the edge lengths are not independent random variables, and, most importantly, the triangle inequality holds. The triangle inequality has implications on our algorithm (for instance, if in a triangle two of the edges are bad, the probability that also the third one is bad is larger than when the lengths are independent).

We start by analyzing the uniform Euclidean instances, which turn out to be the most difficult instances for our algorithm. Let us denote by $U_\Delta(d)$ the instances where each coordinate x_k^i is a uniform random variable in $[0, 1]$. We have the following result:

Theorem 4 *For any constant $d \in \mathbb{N}$, the average running time of $\mathcal{A}_h$ on instances in $U_\Delta(d)$ is $\Theta(n^3)$.*

Proof: The maximum distance between two points in $[0, 1]^d$ is $\sqrt{d}$ and it is achieved by any pair of opposed vertices of the hypercube, i.e., two points $a, b \in \{0, 1\}^d$ such that $a_i = 1 - b_i$ for $i = 1, \ldots, d$. Notice that for each point $p \in [0, 1]^d$ there is at most one point $q \in [0, 1]^d$ such that $||p - q|| = \sqrt{d}$ and therefore for each triangle (i, j, k) it is $\Delta(i, j, k) < 3\sqrt{d}$. Let l be the value of the largest triangle possible in $[0, 1]^d$. Then $l/3 < \sqrt{d}$.

Consider then two open balls, one centered in $(0, \ldots, 0)$ and the other in $(1, \ldots, 1)$, of radius $r = (\sqrt{d} - l/3)/2$ each. Let A and B be the (non-empty) intersections of these balls with the unit cube. Notice that for each $i \in A$ and $j \in B$ it is $||i - j|| > l/3$. Let v be the volume of A (equal to the volume of B). The value v is a constant depending only on d. For each instance in $U_\Delta(d)$, $vn = \Theta(n)$ points are expected to fall in A (respectively, in B) on average. By construction, each edge ij with $i \in A$ and $j \in B$ is good at any stage of the algorithm and hence there are $\Theta(n) \times \Theta(n) = \Theta(n^2)$ expansions for a total of $\Theta(n^3)$ triangles evaluated. □

Notice that, while there are Euclidean instances that require cubic time to be solved, there are also instances which are solved in time $O(n^2)$. One case is when the points are not spread uniformly, but most of them lie within a small-radius ball. The following lemma gives an example.

Lemma 3 *Let k be a constant. Denote by $L_{\max} := \max_{i,j} L_{ij}$. The set of Euclidean instances for which there exists an open ball B of radius $L_{\max}/6$ which contains all points except for at most k points can be solved by $O(n)$ expansions.*

Proof: Notice that a triangle of value $> L_{\max}$ is found at the first expansion. At this point, the only edges which can be expanded are either the $\binom{k}{2} = O(1)$ edges between points out of B, or the $k(n - k) = O(n)$ between a point $i \notin B$ and $j \in B$. □

n	$\binom{n}{3}$	$U_\Delta(2)$	ρ	$U_\Delta(\lceil \log n \rceil)$	ρ	$\tilde{U}_\Delta(2)$	ρ
20	1140	203	-	143	-	135	-
40	9880	925	4.6	542	3.8	513	3.8
80	82160	4716	5.1	2021	3.7	2026	3.9
160	669920	26398	5.6	7372	3.6	8170	4.0
320	5410240	158352	6.0	27844	3.8	33207	4.1
640	43486080	1017851	6.4	99199	3.9	130057	3.9
1280	348706560	6969994	6.8	382551	3.9	536533	4.1
2560	2792926720	49394123	7.1	1422908	3.7	2128656	4.0
5120	22356515840	362727298	7.3	5292483	3.7	8404913	3.9
ratio	-	62×	-	4224×	-	2660×	-

Table 6: Statistics for Euclidean uniform instances.

There is also the possibility of Euclidean instances which require the evaluation of only $O(n)$ triangles (e.g., when there is only one expansion). For example, consider an instance in which 3 points P, Q, R are vertices of an equilateral triangle of side 1, while all other points lie within a circle of radius $(\sqrt{3}-1)/\sqrt{3}$ centered at the triangle's centroid. The triangle PQR is found at the first expansion. Then the threshold for a good arc is set at 1, but there are no edges ij such that $L_{ij} > 1$.

In Table 6 we report the results for Euclidean instances with uniform distribution. The column labeled $U_\Delta(2)$ shows the cubic behavior of the algorithm, which, however, is still much better than complete enumeration with respect to the multiplicative constant. Indeed, the algorithm $\mathcal{A}_h$ evaluates about 1/60th of all triples when $n \simeq 5000$. While for fixed d the algorithm is cubic, we noticed (results not documented here) that the savings with respect to complete enumeration are increasing with d. We have then considered the case in which d is not a constant, but it depends on n. In particular, we have set $d := \lceil \log n \rceil$. The results, reported under the column $U_\Delta(\lceil \log n \rceil)$, show that $\mathcal{A}_h$ performs very well and appears again to be quadratic.

Furthermore, in order to assess the importance of this particular type of Euclidean costs and of the dependance between the variables, we have then performed the following experiment. After having generated n random points, and the corresponding $m := n(n-1)/2$ random lengths $l_1, \ldots, l_m$, we have assigned these lengths randomly to the m edges of the graph. We call this type of instance a *shuffled* Euclidean instance. In column $\tilde{U}_\Delta(2)$ we see the effect of shuffling the edge costs on a Euclidean instance: the algorithm ceases to be cubic and resumes its quadratic behavior.

When the coordinates are generated according to some distribution other than the uniform, the algorithm $\mathcal{A}_h$ performs again very well and shows a quadratic time complexity. In particular, we have considered the Normal, Lognormal, and Beta distributions and have generated the coordinates of each point by using these distributions in turn. The results are reported in Table 7 for var-

ious parameters of the distributions. It is clear from the tables that on all instances the algorithm shows a quadratic, or near-quadratic, complexity. The hardest instances appear to be when the coordinates are generated with the Lognormal distribution for $\mu = 1, \sigma = 2$. Still, in this case the number of triples evaluated for $n \simeq 5000$ is about 1/1200 of the total. The easiest instances appear to be the standard Normal and Lognormal with $\mu = 0, \sigma = 0.1$, which show an almost identical behavior.

n	$\binom{n}{3}$	$N_\Delta(0,1)$	ρ	$LN_\Delta(1,0.1)$	ρ	$LN_\Delta(1,2)$	ρ
20	1140	131	-	130	-	214	-
40	9880	484	3.7	497	3.8	894	4.2
80	82160	1873	3.9	1894	3.8	3727	4.2
160	669920	7074	3.8	7108	3.8	14978	4.0
320	5410240	26362	3.7	27861	3.9	62409	4.2
640	43486080	99278	3.8	107456	3.9	263701	4.2
1280	348706560	389880	3.9	403978	3.8	1074510	4.1
2560	2792926720	1500265	3.8	1574061	3.9	4231876	3.9
5120	22356515840	5788300	3.9	6178092	3.9	17733492	4.2
ratio	-	3862×	-	3617×	-	1261×	-
n	$\binom{n}{3}$	$\beta_\Delta(0.5,2)$	ρ	$\beta_\Delta(2,2)$	ρ	$\beta_\Delta(2,0.5)$	ρ
20	1140	152	-	138	-	152	-
40	9880	557	3.7	519	3.8	562	3.7
80	82160	2034	3.7	1978	3.8	2024	3.6
160	669920	7446	3.7	7424	3.8	7534	3.7
320	5410240	29713	4.0	28299	3.8	29807	4.0
640	43486080	119650	4.0	108981	3.9	121715	4.1
1280	348706560	490836	4.1	441010	4.0	494952	4.1
2560	2792926720	2011397	4.1	1770901	4.0	1941702	3.9
5120	22356515840	8186154	4.1	6918339	3.9	8002915	4.1
ratio	-	2731×	-	3231×	-	2794×	-

Table 7: Statistics for Euclidean instances in $\mathbb{R}^2$ with various distributions of the random coordinates.

8 Conclusions

In this paper we have described and analyzed a deterministic algorithm $\mathcal{A}_h$ for finding the largest triangle in a graph. This problem is believed to be inherently cubic for a worst-case algorithm, and it is subcubic equivalent to many important graph problems. Our algorithm is based on selecting edges and expanding them into triples. By selecting the edges in decreasing order of length and expanding them only if they pass the test of being good, we have shown that $\mathcal{A}_h$ takes on average a quadratic time to find the largest triangle in a

graph of uniformly random edge lengths. This is the best possible complexity for this problem. Besides the theoretical analysis focused on the uniform random costs, we have performed extensive experimental tests which show how our algorithm maintains its optimal, quadratic, complexity over a wide range of random instance classes. One important exception to this optimal behaviour is represented by the class of Euclidean instances, on which the algorithm takes average cubic time. Finally, we have described a randomized algorithm $\mathcal{A}_r$ which also appears to behave very well for this problem and which is much simpler to implement.

A A review of useful probabilistic results

A.1 Order statistics from uniform random variables

We briefly recall some well-known results on order statistics obtained, in particular, from a sample of independent uniform random variables (for a more comprehensive overview on order statistics, see, for example, David and Nagaraja, 2003).

Theorem A1 *Let $X_1, \ldots, X_m$, $m \geq 1$, be independent random variables with common distribution function $F(x)$, $x \in \mathbb{R}$. The distribution function of the maximum $X_{(m)} = \max\{X_1, \ldots, X_m\}$ is $F_{X_{(m)}}(x) = F(x)^m$, $x \in \mathbb{R}$. If the random variables are continuous with density $f(x)$, $x \in \mathbb{R}$, then $X_{(m)}$ has density $f_{X_{(m)}}(x) = mf(x)F(x)^{m-1}$, $x \in \mathbb{R}$.*

Theorem A2 *If the independent random variables $X_1, \ldots, X_m$ follow a continuous uniform distribution with support $[0, 1]$, then $X_{(m)}$ follows a Beta distribution with parameters m and 1 and the associated mean value is $E(X_{(m)}) = m/(m + 1)$. More generally, the k-th order statistic $X_{(k)}$, $k = 1, \ldots, m$, that is the k-th-smallest value in the initial sample, follows a Beta distribution with parameters k and $m + 1 - k$ and $E(X_{(k)}) = k/(m + 1)$.*

Notice that, as expected, $\lim_{m \to +\infty} E(X_{(m)}) = 1$.

A.2 The maximum of triangular random variables

We consider the random variables $X_1, \ldots, X_m$ defined as $X_v = X_{av} + X_{bv}$, $v = 1, \ldots, m$, with $X_{w,v}$, $w = a, b$, $v = 1, \ldots, m$, independent uniform random variables in $[0, 1]$. Since X_v, $v = 1, \ldots, m$, follows a triangular distribution, which is a particular instance of the Irwin-Hall distribution (Hall, 1927; Irwin, 1927), from Theorem A1 we state that the maximum $X_{(m)}$ has density

$$f_{X_{(m)}}(x) = \begin{cases} m\, x\, (x^2/2)^{m-1} & \text{if } x \in [0, 1] \\ m\, (2 - x)\{1 - (2 - x)^2/2\}^{m-1} & \text{if } x \in (1, 2] \\ 0 & \text{if } x \notin [0, 2]. \end{cases}$$

A useful lower bound for the associated expected value is given by the following theorem.

Theorem A3 *The expected value of the maximum of $m \geq 1$ independent triangular random variables is*

$$E(X_{(m)}) = 2 - \frac{2^{-m}}{2m+1} - \int_0^1 (1 - x^2/2)^m dx, \tag{A1}$$

and the following inequality holds

$$E(X_{(m)}) \geq 2 - \frac{2^{-m}}{2m+1} - \sqrt{\pi/(2m)}. \tag{A2}$$

Proof: Since

$$E(X_{(m)}) = \int_0^1 m\, x^2 (x^2/2)^{m-1}\, dx + \int_1^2 m\, x\, (2-x)\{1 - (2-x)^2/2\}^{m-1} dx,$$

using integration by parts, we easily obtain equation (A1). Notice that the integral in (A1) corresponds to a suitable Gaussian or ordinary hypergeometric function (see, for example, Abramowitz and Stegun, 1972). Moreover, since $(1 - x^2/2)^m \leq e^{-mx^2/2}$, $x \in [0, 1]$, we have that

$$\begin{aligned} \int_0^1 (1 - x^2/2)^m dx &\leq \int_0^1 e^{-mx^2/2} dx = \sqrt{2\pi/m} \int_0^1 \sqrt{m}\, \phi(x\sqrt{m}) dx \\ &= \sqrt{2\pi/m}\, \left\{\Phi(\sqrt{m}) - 1/2\right\} \leq \sqrt{\pi/(2m)}, \end{aligned}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ are, respectively, the density and the distribution function of a standard normal random variable. The last inequality follows from the fact that $\Phi(\sqrt{m}) - 1/2 \leq 1/2$, for $m \geq 1$, and that the limit $1/2$ is rapidly attained for moderate values of m. Using this result we immediately obtain the lower bound specified in equation (A2). □

Furthermore, it is quite immediate to prove that, if we introduce an upper boundary condition on the original uniform random variables, the conditional expectation for the maximum $X_{(m)}$ corresponds to the unconditional one multiplied by the boundary level $\xi \in (0, 1)$, namely

$$E(X_{(m)} | X_{wv} \leq \xi, w = a, b, v = 1, \ldots, m) = \xi\, E(X_{(m)}). \tag{A3}$$

A further useful result concerns the expectation of the maximum under the condition that it is itself greater than a suitable lower bound.

Theorem A4 *Let us consider the maximum $X_{(m)}$ of $m \geq 1$ independent triangular random variables. Then, for each $\beta \in (1, 2)$,*

$$\begin{aligned} E(X_{(m)} | X_{(m)} > \beta) \geq\ & 2 - \frac{1}{1 - \{1 - (2-\beta)^2/2\}^m} \left[\sqrt{\frac{\pi}{2m}} \left\{ 2\Phi\left((2-\beta)\sqrt{m}\right) \right.\right. \\ & \left.\left. -1 \right\} + (\beta - 2) \left\{ 1 - \frac{(2-\beta)^2}{2} \right\}^m \right]. \end{aligned} \tag{A4}$$

Proof: Since the conditional density function of $X_{(m)}$ given $X_{(m)} > \beta$ is

$$f_{X_{(m)}|X_{(m)}>\beta}(x) = \begin{cases} m\,(2-x)\{1-(2-x)^2/2\}^{m-1}/P(X_{(m)} > \beta) & \text{if } x \in (\beta, 2] \\ 0 & \text{if } x \notin (\beta, 2], \end{cases}$$

where

$$P(X_{(m)} > \beta) = 1 - \{1-(2-\beta)^2/2\}^m, \tag{A5}$$

the conditional expectation is given by

$$E(X_{(m)}|X_{(m)} > \beta) = \frac{1}{1-\{1-(2-\beta)^2/2\}^m} \int_\beta^2 m\,x\,(2-x)\{1-(2-x)^2/2\}^{m-1} dx.$$

Using the same approach considered before for the unconditional case, we obtain the following lower bound for the conditional expectation

$$\begin{aligned} E(X_{(m)}|X_{(m)} > \beta) &= \frac{1}{1-\{1-(2-\beta)^2/2\}^m} \left[2 - \beta \left\{ 1 - \frac{(2-\beta)^2}{2} \right\}^m \right. \\ &\quad \left. - \int_0^{2-\beta} (1-x^2/2)^m\, dx \right] \\ &\geq \frac{1}{1-\{1-(2-\beta)^2/2\}^m} \left[2 - \beta \left\{ 1 - \frac{(2-\beta)^2}{2} \right\}^m \right. \\ &\quad \left. - \sqrt{\frac{\pi}{2m}} \left\{ 2\Phi\left((2-\beta)\sqrt{m}\right) - 1 \right\} \right]. \end{aligned}$$

The last term on the right hand side corresponds to the lower bound specified in (A4). □

References

[1] A. Abboud, V. Vassilevska-Williams and H. Yu, Matching triangles and basing hardness on an extremely popular conjecture, SIAM Journal on Computing, 47(3), 1098–1122, 2018.

[2] M. Abramowitz and I. A. Stegun, Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, Wiley-Interscience, New York, 1972.

[3] M. de Berg, K. Buchin, B. M. P. Jansen and G. J. Woeginger, Fine-grained complexity analysis of two classic TSP variants, Proceedings 43rd International Colloquium on Automata, Languages and Programming ICALP, 1–14, 2016.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, 3rd Edition, The MIT Press, 2009.

[5] H. A. David and H. N. Nagaraja, H. N., Order Statistics. Wiley, 2003.

[6] D. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In: Aarts, E.H.L. and Lenstra, J.K., Eds., Local Search in Combinatorial Optimization, John Wiley & Sons, New York, 215–310, 1997.

[7] P. Hall, The Distribution of means for samples of size N drawn from a population in which the variate takes values between 0 and 1, all such values being equally probable, Biometrika, 19, 240–245, 1927.

[8] J. O. Irwin, On the frequency distribution of the means of samples from a population having any law of frequency with finite moments, with special reference to Pearson's type II, Biometrika, 19, 225–239, 1927.

[9] G. Lancia and M. Dalpasso, Speeding-up the exploration of the 3-OPT neighborhood for the TSP, in New Trends in Emerging Complex Real Life Problems, AIRO Springer Series 1, 345–356, 2018.

[10] C. Rego, D. Gamboa, F, Glover and C. Osterman, Traveling salesman problem heuristics: Leading methods, implementations and latest advances. European Journal of Operational Research, 211, 427-441, 2011.

[11] G. Reinelt, TSPLIB - A traveling salesman problem library, ORSA Journal on Computing, 3, 376–384, 1991.

[12] V. Vassilevska-Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems, Journal of the ACM, 65(5), 27:1–27:38, 2018.