Bidirectional Dijkstra's Algorithm is Instance-Optimal

Bernhard Haeupler* Richard Hladík† Václav Rozhoň‡ Robert E. Tarjan§
Jakub Tětek¶

Abstract

While Dijkstra's algorithm has near-optimal time complexity for the problem of finding the shortest *st*-path, in practice, other algorithms are often superior on huge graphs. A prominent such example is the *bidirectional search*, which executes Dijkstra's algorithm from both endpoints in parallel and stops when these executions meet.

In this paper, we give a strong theoretical justification for the use of such bidirectional search algorithms. We prove that for weighted multigraphs, both directed and undirected, a careful implementation of bidirectional search is instance-optimal with respect to the number of edges it explores. That is, we prove that no correct algorithm can outperform our implementation of bidirectional search on *any single instance* by more than a constant factor.

For unweighted graphs, we show that bidirectional search is instace-optimal up to a factor of $O(\Delta)$ where Δ is the maximum degree of the graph. We also show that this is the best possible.

1 Introduction

From a theoretical perspective, Dijkstra's algorithm, with its near-linear time complexity, is close to optimal for the problem of finding the shortest path between two vertices s and t. However, when the input graph is huge, we can often find the shortest path between two vertices without exploring the entire input graph. In this scenario, other algorithms are often significantly more efficient in practice than Dijkstra's algorithm. One method that stands out is *bidirectional search*. This approach, proposed by Dantzig [14] in 1963 and Nicholson [37] in 1966, executes Dijkstra's algorithm from both s and t and halts when the two executions meet. While in the worst case, this algorithm has to look at all the nodes and edges of the input graph, it often performs much better in practice.

In this paper, we explain why: On weighted multigraphs with positive weights, a version of this method is *instance-optimal* in terms of the number of edges that the algorithm accesses. This means that the algorithm is the most efficient one for *every single instance*. Concretely, up to a constant factor, there is no correct algorithm that would access fewer edges than bidirectional Dijkstra on even a single input. This result works in the adjacency list model of sublinear algorithms where we assume that we have only a simple query access to the nodes and edges in the graph and we are not given any additional information about it. Note that the time complexity of bidirectional search is proportional to the number of edges accessed, up to a logarithmic term. Thus, bidirectional search is also close to being instance-optimal from the classical time-complexity perspective. We also prove a similar result for the class of unweighted graphs and bidirectional BFS.

Theorem 1.1. (Informal corollary of Theorems 5.1, 6.1 and 6.2) *In the adjacency list model of sublinear algorithms, there exists an instance-optimal algorithm for the shortest st-path problem on weighted multigraphs. On* unweighted *multigraphs, there exists an algorithm that is instance-optimal up to $\Delta(G)$[1]; this factor cannot be improved.*

*bernhard.haeupler@inf.ethz.ch, INSAIT, Sofia University "St. Kliment Ohridski" & ETH Zurich
†rihl@uralyx.cz, INSAIT, Sofia University "St. Kliment Ohridski" & ETH Zurich
‡vaclavrozhon@gmail.com, INSAIT, Sofia University "St. Kliment Ohridski"
§ret@cs.princeton.edu, Princeton University
¶j.tetek@gmail.com, INSAIT, Sofia University "St. Kliment Ohridski"
[1]We use $\Delta(G)$ or just Δ to denote the maximum degree of G.

While the proof of Theorem 1.1 is technically straightforward, we believe that the result is appealing for a number of reasons. First, we stress that the proven guarantee for bidirectional search is extremely strong. The result makes it clear that bidirectional search is the asymptotically best possible algorithm for the shortest st-path problem on *any given input graph*, provided that we are not given any additional information about the input graph.

Second, there are several standard variants of bidirectional search (see Section 2). While we know of three variants proposed in the literature [14, 37, 38], none of them are instance-optimal. To prove Theorem 1.1, we have to use a different very simple variant of the algorithm.

Third, our result expands the set of problems that are now known to be amenable to the strong guarantee of instance optimality. While there are several areas of computer science where the results of this type are known, it seems that in the setup of Theorem 1.1 – sublinear graph algorithms – our result is the first of its kind. We discuss other problems amenable to this type of analysis in Section 2.

Finally, the shortest st-path problem is interesting from the perspective of determining which conditions are necessary for achieving instance optimality. In particular, our results show that certain subtle assumptions about edge weights are crucial. Without these assumptions, instance optimality becomes impossible to attain. This highlights the delicate choices one sometimes has to make to prove instance optimality. We discuss this in Section 5.1.

Optimality of unidirectional Dijkstra's algorithm in a restricted setting Additionally, we prove in Section 4 that in directed graphs, assuming we may access only the out-neighbors of vertices and that we cannot access the degrees, unidirectional Dijkstra's algorithm is instance-optimal. The proof is very simple and it showcases well the general approach that we use in our other proofs. For the sake of simplicity of exposition, we only prove this claim for the class of deterministic algorithms, while we prove our other results also for randomized algorithms.

While the fact that our lower bounds also hold for randomized algorithms may seem uninteresting at first, we think that the opposite is true. Note that while the classical shortest-paths algorithms like Dijkstra's [18] or Bellman-Ford [5, 42] are deterministic, many state-of-the algorithms for the shortest path problem in various models of computation are, in fact, randomized. This includes the fastest known algorithm for computing distance in undirected graphs by Duan, Mao, Shu, and Yin [20] or the state-of-the-art parallel algorithms for the shortest path problem [10, 11, 9, 41]. We also note that the sublinear model of computation is notorious for requiring randomization in order to get any non-trivial algorithm for most problems.

2 Related work

This section presents an overview of related work. First, we explain the bidirectional search meta-algorithm and its variants proposed in the literature. Next, we discuss the connections with the popular A* algorithm, which is used for the same task as bidirectional search in the case we have additional information about the input graph. Finally, we discuss other setups and problems that admit instance-optimality-based analysis.

The bidirectional Dijkstra's algorithm This algorithm was first proposed by Dantzig [14] but his description was very vague. The first to precisely specify a correct algorithm was Nicholson [37]. There are several variants of the bidirectional Dijkstra's algorithm as there is quite significant flexibility in the algorithm. We formulate the core structure of the algorithm as a meta-algorithm in Algorithm 1.

Selection rule Several rules have been proposed for selecting between the two searches in Algorithm 1. Dantzig [14] suggests alternating between exploring a vertex in one of the two executions. On the other hand, Nicholson [37] suggests always exploring the node that is closer to s or t, respectively. Finally, Pohl [38] suggests that we explore in the execution in which the number of vertices that have been seen but not yet explored is smaller. It can be seen that none of those approaches achieve the instance-optimality guarantees of Theorem 1.1. Therefore, in our instance-optimal Algorithm 2 we will use a different, yet very simple equal-work rule in which we simply alternate between relaxing an edge in the forward and backward run.

Stopping condition There are also differences in the stopping condition: Dreyfus [19] suggests to stop when the two executions meet (for a correct definition of "meets"). Pohl [38] suggests a stopping condition that uses distance bounds computed by the algorithm to determine that the shortest path has already been found. In our paper, we use this approach.

One has to be cautious of a seemingly natural stopping condition that is not correct: if we stop when the two

Algorithm 1: Bidirectional Search Meta-Algorithm

Input: Graph $G(V, E)$, source vertex s, target vertex t
Input: Selection function, stopping condition
Initialize forward search from s and backward search from t;
while *stopping condition is not satisfied* **do**
  Use a selection rule to select one of the two directions;
  if *direction is forward* **then**
    Execute one edge relaxation of Dijkstra's algorithm from s;
  else
    Execute one edge relaxation of reverse Dijkstra's algorithm from t;
  end
end
Recover the shortest path from the two runs;

sets of open vertices intersect for the first time, the vertex at which the two executions meet may not lie on the shortest path. Indeed, this incorrect stopping rule appeared in the literature as was pointed out by Pohl [38].

Optimality We are not aware of works that give theoretical guarantees of bidirectional search comparable to Theorem 1.1. In [38], Pohl gives a heuristic argument. Namely, he models the input graph in a continuous probabilistic way and argues heuristically that under this model, his version of bidirectional search is the best possible. In general, it is known that bidirectional search or algorithms based on it can have sublinear time complexity for certain specific classes of graphs such as hyperbolic random graphs [6], power-law graphs [8], Chung-Lu random graphs [4], or expanding graphs [2, 7].

Relation to the A* algorithm One might wonder how it can be that bidirectional Dijkstra is instance-optimal, but in practice, it is often outperformed by the famous A* algorithm. The reason is that A* requires additional advice (heuristical distance estimates) as a part of the input. More formally, A* solves a different problem than bidirectional Dijkstra; specifically, a problem where the input consists of both the graph and the additional advice on every node. This shows an important nuance of Theorem 1.1: bidirectional search is optimal only if we assume that we have no way of learning additional meaningful information about the input graph.

The A* algorithm was first suggested by Hart, Nilsson, and Raphael [30]. The algorithm was later shown to be optimal among all unidirectional-Dijkstra-like algorithms by Dechter and Pearl [15].

A bidirectional version of the A* algorithm was suggested by Holte, Felner, Sharon, Sturtevant, and Chen [31]. Different versions of the bidirectional A* algorithm have been studied; we refer the interested reader to [31, 44, 43]. Eckerle, Chen, Sturtevant, Zilles, and Holte [21] made significant progress in proving an analogous result that bidirectional A* is optimal among bidirectional-Dijkstra-like algorithms. Shaham, Felner, Sturtevant, and Rosenschein [43] in fact used these results to give algorithms that are optimal among all bidirectional-Dijkstra-like algorithms, if one sets optimally a parameter of that algorithm. However, one does not know the optimal parameter in advance, meaning that this result falls short of giving an algorithm optimal among bidirectional-Dijkstra-like algorithms.

While our techniques are straightforward and similar to these papers, our Theorem 1.1 is significantly stronger because we prove optimality among *all* correct algorithms, while the previous work [21, 43] restricts itself to certain classes of algorithms. Moreover, we show optimality in terms of the number of edge accesses, not vertex accesses. Since the time complexity is near-linear in the number of edge accesses of the algorithm, this means that our algorithm is also near-instance-optimal in terms of its actual time complexity, and not just the number of encountered nodes.

Instance optimality for other problems Due to its strength, instance optimality is an extremely appealing beyond-worst-case guarantee an algorithm can have; see the relevant chapter in Roughgarden [39] for an introduction. There are several known setups where we can make instance-optimality-based analyses of algorithms work.

The original paper by Fagin, Lotem, and Naor [22] that coined the term comes from the area of greedy algorithms for retrieving data from databases.

Several algorithms with instance-optimality-like guarantees are known for problems related to sorting, such as the problem of finding the convex hull [1], computing set intersections [16], sorting with partial information [26, 32], or sorting vertices of an input graph by their distance (the problem solved by Dijkstra's algorithm) [27].

In the area of distributed algorithms, many algorithms can be proved to have a guarantee called universal optimality that is closely connected to instance optimality [23, 28, 24]. This includes algorithms for the approximate shortest path problem [29, 47, 40].

Another area where instance-optimal algorithms are known is the area of sequential estimation [46, 45, 33]. There is an instance-optimal sublinear-time algorithm for computing the distance of a point from a given curve [3]. Yet another area with known instance-optimal algorithms is that of multi-armed bandits, with works on that topic including [35, 12, 13, 36, 34].

3 Preliminaries

In this section, we review standard concepts necessary for the later analysis.

Our model of sublinear algorithms We work within a standard query model for sublinear graph algorithms [25, Chapter 10]. The complexity measure of interest is then the number of queries performed. We assume that the input contains a weighted multigraph G with n nodes; the edges have real weights. Note that a multigraph can contain both self-loops and parallel edges.

We assume that the graph G is stored in the adjacency list format. That is, we have query access to its vertices, each of which keeps a list of its neighbors. More formally, vertices are numbered 1 to n and on undirected graphs, we are allowed to perform the following queries:

1. **Degree**(i): Given an input identifier i with $1 \leq i \leq n$, this function returns the degree of the i-th vertex.
2. **Neighbor**(i, j): Given an input identifier i with $1 \leq i \leq n$ and the index of its neighbor j with $1 \leq j \leq \deg_G(i)$[2], this function returns the identifier of the j-th neighbor of i. Additionally, it returns the positive real weight of the edge.

On directed graphs, we can similarly query the **Indegree** and the **Outdegree** of a vertex; moreover, we have functions **Inneighbor** and **Outneighbor** that can list the neighbors of a given vertex in both directions.[3]

Each query has a unit cost and we use the number of queries as our main measure of algorithmic complexity. This measure of *query complexity* is closely related to the classical time complexity since classical algorithms based on Dijkstra's algorithm have their time complexity proportional to the number of edges explored, up to a logarithmic factor necessary for heap maintenance.

We will prove instance optimality in the more general setting of randomized algorithms. For such algorithms, we measure the expected number of queries. A randomized algorithm is said to be correct if it is correct on every instance with a probability at least 0.9.

The shortest *st*-path problem In the shortest *st*-path problem, we are given an input (directed or undirected) multigraph G. The edges of the graph have weights given by function $\ell : E(G) \to \mathbb{R}_{>0}$; this function induces a distance function $d : V(G) \times V(G) \to \mathbb{R}_{\geq 0}$ that maps any two different nodes u, v to their positive shortest-path distance $d(u, v)$ (that might be different from $d(v, u)$ on directed graphs). It is important to consider edges with strictly positive, instead of nonnegative, weights. We discuss the difference later in Section 5.1.

For the shortest *st*-path problem, we are moreover given two vertices s and t of G. The task is to return an arbitrary shortest *st*-path. Note that in the case of multi-graphs, it does not suffice to just output the sequence of vertices but one must output the specific edges.

Dijkstra's algorithm We now recall Dijkstra's algorithm [17] and define terms that we will later use in our proofs. Throughout the execution, we store for each vertex w the length $\hat{d}(s, w)$ of the shortest sw-path found so far. The algorithm starts with the vertex s being *open* and all others being *unvisited*. It then repeatedly takes the vertex u closest to s among all open vertices and *relaxes* all edges uv leaving this vertex as follows. Relaxing an edge to an unvisited vertex v makes v open and sets $\hat{d}(s, v) := \hat{d}(s, u) + \ell(uv)$. If v is open, we update the length of the currently shortest path to v by setting $\hat{d}(s, v) := \min(\hat{d}(s, v), \hat{d}(s, u) + \ell(uv))$. If v is closed, we do nothing.

[2]We use $\deg_G$ to denote the degree in the graph G.

[3]We remark that the instance optimality can also be proven in slightly stronger models allowing additional queries. One such query is **Edges**(i_1, i_2, j) which returns the j-th multiedge between the two nodes i_1, i_2.

Once we have relaxed all edges leaving a vertex, the vertex is *closed* from that point on, and we continue with the next vertex in the list of open vertices. It should be noted that the names unlabeled, labeled, and scanned are also used for unvisited, open, and closed, respectively.

The classical Dijkstra's algorithm finishes once the target node t becomes closed, or when the set of open nodes becomes empty (in which case t is unreachable). However, in bidirectional search, the stopping condition is more complex as there are two runs of Dijkstra's algorithm involved.

Once the algorithm finishes, one can recover the shortest path to any closed vertex v. Namely, we start with v and we find a vertex v' such that $\hat{d}(s,v') = \hat{d}(s,v) - \ell(v',v)$. By iteratively finding the preceding vertex like this, we may recover the whole shortest sv-path.

Instance optimality Intuitively speaking, an algorithm is instance-optimal (up to c) if it is optimal (up to c) on every single instance among the set of correct algorithms.

We say that an *algorithm A is correct* if on any input, it outputs a correct output with probability ≥ 0.9. Note that by standard probability amplification, the constant 0.9 is arbitrary and any constant $> 1/2$ would work.

DEFINITION 1. *A correct algorithm A is instance-optimal under complexity function T if there exists $c = O(1)$ such that on every input x it holds for every correct algorithm A' that the expected complexity $T_A(x)$ and $T_{A'}(x)$ of respectively A and A' on x satisfy*

$$T_A(x) \leq c \cdot T_{A'}(x)\,.$$

The definition of instance optimality readily generalizes to algorithms that are instance-optimal up to a potentially nonconstant factor, such as $\Delta(G)$ that we encountered in Theorem 1.1.

Throughout the paper, we will assume T to be the query complexity. However, note that the time complexity can be off only by a factor of at most $O(\log n)$ required to handle the heap operations necessary in Dijkstra's algorithm.

4 Warm-up: Unidirectional Search in Directed Graphs

In this section, we show the simplest result of the kind that we focus on. Namely, we consider a more restricted query model, deterministic algorithms, and we focus on only computing the distance from s to t, instead of actually finding the path. We then show that standard Dijkstra's algorithm is instance-optimal if aborted at the right time. Unlike in other sections, we also phrase the result here in a self-contained way to make it more accessible to a casual reader.

The proof is conceptually similar to other proofs in this paper, which however need to take care of several additional obstacles. We hope this proof may serve as a warm-up for the other proofs.

THEOREM 4.1. *Let us have a directed weighted graph G with positive weights and assume we are given two vertices s,t. Assume the only operation we can do is to take a vertex we have seen and ask for its next out-neighbor (in an adversarial ordering) and the weight of the edge to that vertex.*

Consider executing Dijkstra's algorithm from s and aborting it once we close some vertex v with $\hat{d}(s,v) = \hat{d}(s,t)$ (possibly $v = t$). Then this algorithm correctly computes the st-distance. Furthermore, no correct deterministic algorithm A can perform fewer queries on G.

Proof. First, we argue correctness. By the standard proof of correctness of Dijkstra's algorithm, once we close the vertex v, we have $\hat{d}(s,v) = d(s,v)$. At the same time, vertices are closed in order of non-decreasing distance, meaning that $d(s,t) \geq d(s,v) = \hat{d}(s,v) = \hat{d}(s,t)$. Moreover, it always holds that $\hat{d}(s,t) \geq d(s,t)$. Thus, we have $\hat{d}(s,t) = d(s,t)$, meaning that the distance is correct.

For the sake of contradiction, let us have an algorithm that performs fewer queries than Dijkstra on G. Therefore, there has to be an edge uv for $d(s,u) < d(s,t)$ that A does not query. We define a graph G' where we replace the edge uv by ut with weight $\delta < d(s,t) - d(s,u)$. The distance between s and t in G' is then $d(s,u) + \delta < d(s,t)$. However, the algorithm does not query this edge. Since the rest of the graph is exactly the same, the algorithm thus returns the same answer on both G and G' which implies that the algorithm is not correct. ∎

5 Instance Optimality in Weighted Graphs

Now we give an instantiation of the bidirectional search meta-algorithm that we later prove is instance-optimal. As the selection rule that chooses between the two executions, we use the perhaps simplest possible rule in which

we alternate one edge relaxation in the forward algorithm with one edge relaxation of the backward algorithm. This way, we make sure that at any point in time, each of the two executions has the same amount of work being invested in it.

As the stopping condition, we use the classical stopping condition of Pohl [38] in Line 18. That is, we are keeping track of the length μ of the currently shortest found path. Moreover, we are keeping track of the values $d(s,u_s), d(u_t,t)$: distances of the vertices that are currently being explored from s and t. Once we know that $d(s,u_s) + d(u_t,t) \geq \mu$, we may terminate our search since all the paths between s and t we have not considered yet have to consist of two disjoint parts of lengths at least $d(s,u_s)$ and at least $d(u_t,t)$.

We present the algorithm formally in Algorithm 2.

Algorithm 2: Instance-Optimal Bidirectional Dijkstra's Algorithm

Input: Graph $G(V,E)$, source vertex s, target vertex t

1 $\mu \leftarrow +\infty$; `// Length of the shortest path that we have found so far`
2 $e_{\text{mid}} \leftarrow \perp$; `// Middle edge of the shortest path that we have found so far`
3 $u_s \leftarrow s, u_t \leftarrow t$; `// Vertices currently being explored in the two executions`

4 Initialize forward search from s on G and backward search from t on G with edges reversed;
5 **while** *neither of the two executions has terminated* **do**
6 | Alternate between relaxing one edge in the Forward and Backward algorithm;
7 **end**

8 $uv \leftarrow e_{\text{mid}}$;
9 $P \leftarrow$ "shortest su-path according to forward execution" + e_{mid} + "shortest vt-path according to backward execution";
10 **return** P;

11 **Function** `Forward_algorithm`**:**
12 | $\hat{d}(s,\cdot) \leftarrow +\infty$; $\hat{d}(s,s) \leftarrow 0$;
13 | Open s;
14 | **while** *an open vertex exists* **do**
15 | | Let u be the open vertex with the smallest $\hat{d}(s,u)$;
16 | | Close u;
17 | | Set $u_s \leftarrow u$;
18 | | **if** $\hat{d}(s,u_s) + \hat{d}(u_t,t) \geq \mu$ **then**
19 | | | **terminate** the whole algorithm;
20 | | **end**
21 | | **for** v *a forward neighbor of* u **do**
22 | | | **if** v *is not closed* **then**
23 | | | | $\hat{d}(s,v) \leftarrow \min(\hat{d}(s,v), \hat{d}(s,u) + \ell(uv))$;
24 | | | **end**
25 | | | **if** v *is closed in the backward execution* **then**
26 | | | | $\mu \leftarrow \min(\mu, \hat{d}(s,u) + \ell(uv) + \hat{d}(v,t))$;
27 | | | | $e_{\text{mid}} \leftarrow uv$
28 | | | **end**
29 | | **end**
30 | **end**
31 **Function** `Backward_algorithm`**:**
32 | Analogous to Forward algorithm with the roles of *forward* and *backward* flipped;

THEOREM 5.1. *Algorithm 2 is an instance-optimal algorithm, under query complexity, for the shortest st-path problem in both directed and undirected graphs with positive weights.*

We prove the theorem next. The proof is conceptually similar to that of Theorem 4.1, but needs to handle some additional issues.

Before the proof, let us recapitulate what we need to prove. First, we need to prove that the algorithm is correct. Then, by Definition 1, we need to show that for any algorithm A' that is correct with probability 0.9, any input (directed or undirected) graph G, and any two vertices s, t, we have that the expected query complexity is $T_{\text{Alg2}}(G, s, t) = O\left(T_{A'}(G, s, t)\right)$.

Since our Algorithm 2 is slightly different than the variants of the bidirectional search that appeared before, we also need to verify its correctness.

Proof. **Correctness.** By correctness of Dijkstra's algorithm, the calculated shortest-path distance $\hat{d}(s, v)$ to any closed vertex v equals the true distance $d(s, v)$. Let d_s, d_t be the respective values of $d(s, u_s)$ and $d(u_t, t)$ when we stopped. The value of μ corresponds to the length of a (not necessarily simple) st-path. We will show it is in fact a shortest st-path. Suppose there is an st-path with length $\mu' < \mu$. There has to be an edge uv on this path such that $d(s, u) \leq d_s$ and $d(v, t) \leq d_t$, and at least one of those two inequalities is strict. Without loss of generality, assume that $d(s, u) < d_s$. Of all such edges uv, pick the one which is in the order on the path the closest to t.

We claim that the other inequality has to also be strict: $d(v, t) < d_t$. Since the algorithm has finished, we know that $d(s, u_s) + d(u_t, t) = \hat{d}(s, u_s) + \hat{d}(u_t, t) \geq \mu$. At the same time, we are assuming that $d(s, u) + \ell(uv) + d(v, t) = \mu' < \mu$. Assume for the sake of contradiction that $d(v, t) = d_t$. We then have

$$d_s + d_t > d(s, u) + \ell(uv) + d(v, t)$$

and thus by our assumption

$$d_s + d_t > d(s, u) + \ell(uv) + d_t\,.$$

But this means that $d_s > d(s, u) + \ell(uv)$. This is in contradiction with our choice of uv as the edge closest to t on the path that satisfies $d(s, u) < d_s$ and $d(v, t) \leq d_t$.

Since $d(s, u) < d_s$ and $d(v, t) < d_t$, the vertex u is closed in the forward execution and v is closed in the backward execution. Consider the one of u, v that has been closed later. When closing this vertex, we also explored uv. We updated μ to $d(s, u) + \ell(uv) + d(v, t) = \mu' < \mu$ on Line 26, a contradiction with the assumption that we returned a path of length μ.

Instance optimality. We seek to prove that no algorithm for the shortest st-path problem can use fewer queries than Algorithm 2 by more than a constant factor. In the following argument, we argue that no algorithm can be faster for the problem of computing the s-t distance (instead of the shortest st-path problem). This implies the claim because any algorithm that computes the path can be easily modified to also calculate the distance without increasing its complexity.

Let E_s and E_t be the sets of the edges that our algorithm explored using the two executions of Dijkstra's algorithm. Note that we have $|E_t| \leq |E_s| \leq |E_t| + 1$ by the definition of Algorithm 2.

First, note that the query complexity of Algorithm 2 is proportional to $|E_s| + |E_t| \leq 2|E_s|$. On the other hand, suppose that an algorithm A explores at most $|E_t|/4$ edges in expectation; here, we say that A explores an edge if it queries it via the operation **Neighbor** described in Section 3 from any of its endpoints. We will prove that under this assumption, the algorithm A is incorrect with a probability of at least $1/2$.

To see this, note that in both E_s and E_t, there exists an edge that is accessed with probability at most $1/4$ by A: Otherwise, by the linearity of expectation, the expected number of edges visited by A would be more than $|E_t|/4$ which we assume is not the case. We call these two edges $e_1 = u_1v_1$ and $e_2 = u_2v_2$ (note that it could happen that $e_1 = e_2$). In the undirected case, we without loss of generality assume that $d(s, u_1) \leq d(s, v_1)$ and $d(v_2, t) \leq d(u_2, t)$.

We next claim that the shortest st-path has its length strictly larger than $d(s, u_1) + d(v_2, t)$. To see this, consider the point in time right before the condition $\hat{d}(s, u_s) + \hat{d}(u_t, t) \geq \mu$ starts being true. At this point, both e_1 and e_2 have already been accessed. We claim that u_1 is closed in the forward execution. In the directed case, this is easy to see, since any edge uv is only ever accessed after u is closed. In the undirected case, either v_1 is not closed and then u_1 must be closed by the same argument, or v_1 is closed and then u_1 must be also closed since we are assuming $d(s, u_1) \leq d(s, v_1)$. Thus, we conclude that u_1 is closed in the forward execution, and analogously, v_2 is closed in the backward execution. Now, since u_s and u_t are the most recently closed vertices in their respective executions, we have $d(s, u_1) \leq d(s, u_s)$ and $d(v_2, t) \leq d(u_t, t)$. Thus, $d(s, u_1) + d(v_2, t) \leq d(s, u_s) + d(u_t, t) = \hat{d}(s, u_s) + \hat{d}(u_t, t) < \mu$.

Next, let us define $\delta = d(s,t) - d(s,u_1) - d(v_2,t)$; note that $\delta > 0$ by our previous claim. We now construct a graph G' with a different distance between s and t than in G with the property that A returns the same answer on both G and G' with large probability.

We start with the case $e_1 \neq e_2$. In this case, we define G' by replacing e_1, e_2 in G by two edges u_1v_2 and v_1u_2 where the first edge has length δ' for arbitrary $0 < \delta' < \delta$ and the second edge has arbitrary weight. Note that the degrees of all vertices are the same in the two graphs, and they also have the same number of vertices, although G' may contain parallel edges and self-loops even if G did not.

Furthermore, note that the only queries that would return different answers on the two graphs are the neighborhood queries that would access either of the edges e_1, e_2 on G but they would access the edges u_1v_2, v_1u_2 on G'. However, we assumed that with probability at least $1/2$, A accesses neither e_1 nor e_2 when run on G. This also implies that A does not access the edges u_1v_2, v_1u_2 on G' with that probability, since until those edges are accessed, the runs of A on G and G' perform the exact same queries (assuming they use same source of randomness).

We conclude that with probability at least $1/2$, A returns the same answer on both graphs G and G'. We will next argue that the correct answers are different on the two graphs.

To see this, we will verify that $d_{G'}(s,u_1) \leq d_G(s,u_1)$ and $d_{G'}(v_2,t) \leq d_G(v_2,t)$.[4] We show the first inequality since the second inequality can be proven in the same way. Specifically, we will prove that no shortest su_1-path uses e_1 or e_2, which implies the desired inequality. We will assume that G is undirected as the directed case is similar and, in fact, easier.

We start with e_1. We recall our assumption that $d(s,u_1) \leq d(s,v_1)$. Together with the fact that all edge weights are positive, this implies that no shortest su_1-path can use the edge e_1 in the direction from v_1 to u_1. The edge is clearly not used in the opposite direction.

We continue with e_2. If it lied on the shortest path from s to u_1, we would have $d(s,v_2) \leq d(s,u_1)$. Moreover, there would be a path from s to t of length at most $d(s,v_2) + d(v_2,t) \leq d(s,u_1) + d(v_2,t)$. However, we have proven that the distance $d(s,t)$ is strictly larger than this quantity.

We conclude that $d_{G'}(s,u_1) \leq d_G(s,u_1)$. Finally, we recall the equality $d_G(s,t) = d_G(s,u_1) + d_G(v_2,t) + \delta$ to conclude that

$$d_{G'}(s,t) \leq d_{G'}(s,u_1) + d_{G'}(v_2,t) + \delta' < d_G(s,u_1) + d_G(v_2,t) + \delta = d_G(s,t).$$

In particular, $d_G(s,t) \neq d_{G'}(s,t)$ and we conclude that A is incorrect with probability at least $1/2$ on either G or G', a contradiction.

Finally, it remains to consider the easier case that $e_1 = e_2$. In this case, the only difference between G and G' is that we set the length of $e_1 = e_2$ in G' to be δ' for arbitrary $0 < \delta' < \delta$. We then have

$$d_{G'}(s,t) \leq d(s,u_1) + \delta' + d(v_2,t) < d(s,t)\,.$$

By an analogous argument as above, it can be argued that A is incorrect with probability at least $1/4$ on either G or G'. □

5.1 Remarks regarding our setup We make a few remarks regarding our setup and our proof of instance optimality.

Flexibility in our algorithm First, we note that there is flexibility in Algorithm 2: instead of alternating between edges, it suffices to make sure that the total number of edges explored in either execution is the same, up to constant factors.

Zero-length edges Next, we remark that it is crucial that we assume that edge-weights are positive reals. In the case when we allow zero-length edges, instance optimality is no longer possible: Just imagine a large graph where all the edges have weight zero. An algorithm taylored to a particular graph G can start with an advice constituting of a path between s and t; it suffices to walk along that path and confirm that $d(s,t) = 0$ while algorithms without such advice have to dutifully explore G.

Formally, nonexistence of instance-optimal algorithms in the case when 0-weight edges are allowed can be proven by considering the graph G_0 that consists of two complete binary trees rooted at s and t. Moreover, select random i and connect by an edge the i-th vertices on the last layer of the two binary trees. One can notice that

[4] In fact, it holds that those distances are the same, but we will not need to argue this.

any algorithm has to make $\Omega(n)$ queries in expectation on a randomly sampled G_0, while for any sample, there exists a fixed fast algorithm tailored to it that only makes $O(\log n)$ queries before finishing.[5]

Positive lower bound on edge-weights It is also crucial that as weights, we allow arbitrarily small positive numbers: In the proof, we define a certain quantity δ and construct a graph G' with edge-weight $0 < \delta' < \delta$. This assumption is again necessary; we will see in Section 6 that even on unweighted graphs (where there is a minimum edge-weight of 1), instance optimality is achievable only approximately. We note that even in the unweighted setting, we could prove that bidirectional search is instance-optimal, if we changed the task of finding the st-distance (or one shortest st-path) to the task of finding *all* shortest st-paths.

Accessing directed edges from both endpoints In our proof, it is also crucial that each (directed) edge can be accessed not only from the source, but also from its target. If it could be accessed only from the source, one could see that the standard Dijkstra's algorithm is instance-optimal (under the assumption of strictly positive real edge-weights).

Computing s-t distance vs computing a shortest st-path Our proof actually shows a somewhat stronger statement. Namely, it show that no algorithm for the easier problem of computing the s-t distance can be faster on any instance than Algorithm 2, which solves the harder problem of computing an actual shortest path.

6 Approximate Instance Optimality in Unweighted Graphs

In this section, we prove that the bidirectional search is approximately instance-optimal also on unweighted graphs. See Figure 1 for the intuition behind the $O(\Delta)$ term that we have to lose in the analysis. We will use the same algorithm, Algorithm 2, for the proof. We note that in the unweighted case, no heap is needed for the implementation and the query complexity and the time complexity are equal. We also call this adapted algorithm *the bidirectional BFS* algorithm.[6]

THEOREM 6.1. *The bidirectional BFS algorithm for the shortest st-path problem in unweighted graphs is instance-optimal, under both query and time complexity, up to the factor of $O(\Delta)$.*

We note that the following proof mostly follows the proof of Theorem 5.1.

Proof. We have already shown correctness in Theorem 5.1. It thus remains to prove the instance-optimality. Just like in the proof of Theorem 5.1, we prove that no faster algorithm can be correct for the easier problem of computing the distance from s to t. Just like in that proof, this implies the claim. We note that in the unweighted setting, once a node u is opened with some value $\hat{d}(s,u)$, we know that $\hat{d}(s,u) = d(s,u)$. We may thus use $\hat{d}$ and d values interchangeably.

Consider the step in Algorithm 2 in which we redefined the value μ to its final value (i.e., the length of the shortest path) for the first time. We will use $\mu^0 = d(s,t)$ to denote this final value. Without loss of generality, we assume that this happened during the forward run.

We use u^0, v^0, w^0 to denote the following three nodes. The node u^0 is the node that is being explored during the step in the forward run that defined μ^0. The node v^0 is its neighbor that was used to define μ^0 as $\mu^0 = \hat{d}(s,u^0) + \ell(u^0,v^0) + \hat{d}(v^0,t)$. Finally, w^0 is the node such that v^0 was opened by w^0 in the backward run.

We will use d_s, d_t to denote the distances $d(s,u^0)$ and $d(w^0,t)$. Note that we have

$$\mu^0 = d_s + 2 + d_t$$

where the additive term $+2$ is for the two edges u^0v^0 and v^0w^0.

We let E_s be the set of edges that are outgoing from any vertex u with $d(s,u) \leq d_s$ and let E_t be the set of edges ingoing to any u such that $d(u,t) \leq d_t$; in the undirected case, replace "outgoing/ingoing" by "adjacent".

For the sake of contradiction, let us assume the existence of an algorithm A' that, in expectation, explores at most $\min(|E_s|, |E_t|)/4$ edges. Then, using the linearity of expectation, we conclude that there are two edges

[5]This construction implicitly assumes that vertices have unique identifiers – otherwise we cannot construct the suitable advice. However, the need for unique identifiers can be easily removed by adding to each vertex a neighbor (center of a star) with degree equal to the identifier.

[6]We note that Algorithm 2 could be further sped up if we also terminate the search once we encounter an edge uv where u was seen in the execution from s and v from t. However, this more practical algorithm is not $o(\Delta)$-instance-optimal, so we focus on the conceptually simpler Algorithm 2.

$e_1 = u_1v_1 \in E_s$ and $e_2 = u_2v_2 \in E_t$, such that with probability at least $1/2$, neither edge is queried by A'. In the undirected case, we assume without loss of generality that $d(s, u_1) \leq d(s, v_1)$ and similarly that $d(v_2, t) \leq d(u_2, t)$. This implies $d(s, u_1) \leq d_s$ and $d(v_2, t) \leq d_t$.

We observe that $e_1 \neq e_2$. Otherwise, we would have that

$$d(s,t) \leq d(s,u_1) + 1 + d(v_1,t) \leq d_s + 1 + d_t$$

which would be in contradiction with $d(s,t) = d_s + 2 + d_t$.

We define a new multigraph G' by replacing the edges u_1v_1, v_2u_2 by u_1u_2 and v_1v_2. Note that G' may contain self-loops and parallel edges even if G did not contain those. We observe that the only queries that distinguish G from G' correspond to accessing u_1v_1 or u_2v_2 in G. Thus, A behaves differently on G and G' with probability at most $1/2$. We argue below that $d_G(s,t) \neq d_{G'}(s,t)$; this implies that the success probability of A is at most $1/2$ which in turn proves that any correct algorithm has to have query complexity at least $\Omega(\min(|E_s|, |E_t|))$ on G.

We now need to argue that $d_G(s,t) \neq d_{G'}(s,t)$. We first claim that $d_{G'}(s,u_1) \leq d_G(s,u_1)$ and $d_{G'}(v_2,t) \leq d_G(v_2,t)$; we will argue only for the first inequality as the second proof is the same. Specifically, we will prove that no shortest su_1-path uses e_1 and e_2 which implies our claim since then any shortest su_1-path in G also exists in G'. We will assume that G is undirected as the directed case is similar and, in fact, easier.

We start with e_1. We recall our assumption that $d(s,u_1) \leq d(s,v_1)$. This implies that any su_1-path that uses the edge e_1 in the direction from v_1 to u_1 has length at least $d(s,v_1) + 1 \geq d(s,u_1) + 1$ and is thus not a shortest path. The edge is clearly not used in the opposite direction.

We continue with e_2. If it lied on a shortest path from s to u_1, we would have $d(s,v_2) \leq d(s,u_1) - 1 \leq d_s - 1$. Moreover, we have $d(v_2,t) \leq d_t$. This implies $d(s,t) \leq d(s,v_2) + d(v_2,t) \leq d_s + d_t - 1$ which is in contradiction with the fact that $d(s,t) = d_s + d_t + 2$.

We conclude that $d_{G'}(s,u_1) \leq d_G(s,u_1)$ and analogously $d_{G'}(v_2,t) \leq d_G(v_2,t)$. We can now use this to compute that

$$\begin{aligned} d_{G'}(s,t) &\leq d_{G'}(s,u_1) + d_{G'}(v_2,t) + 1 \\ &\leq d_G(s,u_1) + d_G(v_2,t) + 1 \\ &\leq d_s + d_t + 1\,. \end{aligned}$$

On the other hand, we have by definition that

$$d_G(s,t) = \mu = d_s + d_t + 2$$

and we thus have $d_G(s,t) \neq d_{G'}(s,t)$.

It remains to argue that the time complexity of the algorithm is at most $O(\Delta \cdot \min(|E_s|, |E_t|))$. To see this, we first observe that the forward search in our algorithm closes at most one vertex of distance at least $d_s + 2$ from s. At the point in time when such a node u is first closed, we have $\hat{d}(s,u_s) \geq d_s + 2$. However, we would also have $\hat{d}(u_t,t) \geq d_t$ since the node w^0 was closed by the backward algorithm, by definition of w^0. Thus, we have $\hat{d}(s,u_s) + \hat{d}(u_t,t) \geq d_s + d_t + 2 = \mu^0$; this however triggers the condition on Line 18 and the algorithm terminates. We conclude that our algorithm closes at most $|E_s| + 1$ vertices and hence it explores $O(|E_s| \cdot \Delta)$ edges. An analogous argument can be made for the backward search. We conclude that our algorithm makes $O(\min(|E_s|, |E_t|) \cdot \Delta)$ queries, as needed. ∎

6.1 Lower bound Next, we prove that the factor of $O(\Delta)$ in Theorem 6.1 cannot be improved. We prove this in a slightly more general setup where the set of allowed positive weights, $\mathcal{W}$, is bounded away from zero.

THEOREM 6.2. *Assume the shortest st-path problem when the allowed graph weights come from a set $\mathcal{W}$ with $\nu := \min(\mathcal{W}) > 0$ and we restrict the class of input graphs to those of degree at most Δ. Then there is no algorithm that is instance optimal, under both query and time complexity, for the problem up to a factor of $o(\Delta)$.*[7]

[7]We do not regard Δ as a constant in the o notation.

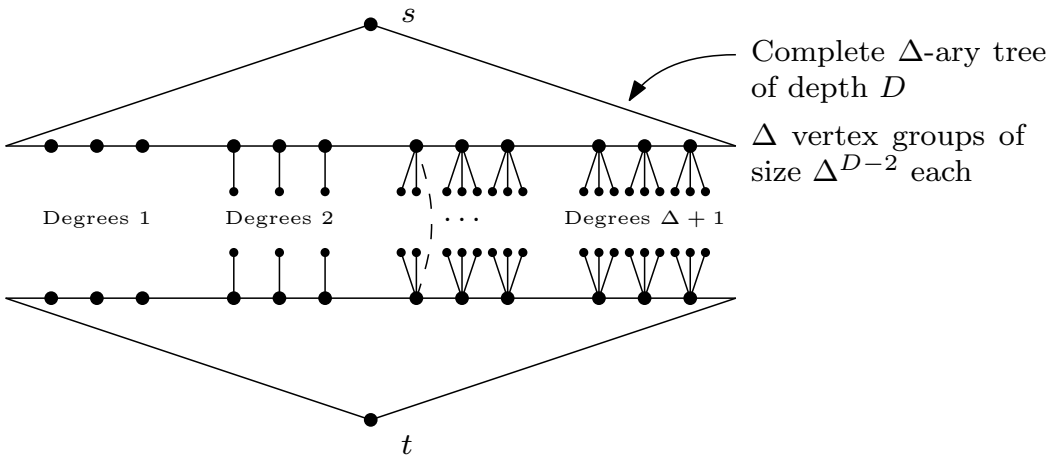

Figure 1: The lower bound instance from Theorem 6.2: There are Δ groups of vertices and one of them contains a hidden edge connecting s to t. If we know which group contains the hidden edge, we only need to search one of the groups, otherwise we have to search all of them. Note that this counterexample crucially uses that the hidden edge has the smallest possible weight; otherwise, all the groups need to be searched as they could contain an edge with even smaller weight.

Proof. Consider the following graph G_1 illustrated in Figure 1. All edges in G_1 have weight ν. To construct G_1, we first construct a tree as follows. Consider a complete Δ-ary tree of depth D rooted in s – for convenience, we use Δ to be the arity of the tree, resulting in maximum degree of $\Delta + 1$. Remove some of the vertices in the last layer so that the first Δ^{D-2} vertices on the penultimate layer have degree $\Delta + 1$, the next Δ^{D-2} vertices have degree Δ, and so on. We then construct a second, isomorphic tree rooted in t. Next, for some value i, we consider the i-th edge in the last layer in both of these trees, we remove the two edges, and connect the corresponding two vertices on the penultimate layer with a new edge.

If i is chosen uniformly randomly, any algorithm with a constant success probability has to visit, in expectation, a constant fraction of all vertices on the last level before it finds the connecting edge. Hence, its expected complexity is $\Omega(\Delta^D)$. On the other hand, for any fixed graph, we will now describe an algorithm that is correct and runs in time $O(\Delta^{D-1})$.

Consider an instance of the graph G_1 where the connecting edge connects vertices with degree k. Next, consider an algorithm that first runs a bidirectional BFS taylored to G_1 and k that works as follows. The algorithm explores the vertices from s and t until the penultimate layer. Then, it explores the vertices with degree k. If it finds an edge of length ν between the two components containing s and t, the algorithm finishes. If such an edge is not found, or if the algorithm finds that the input graph is not G_1, the algorithm disposes of the run and runs any correct algorithm such as Dijkstra's algorithm to compute the result.

We claim that this algorithm is correct on all inputs. This follows from the fact that once the two subtrees are explored, any shortest path has to have length at least $(2D-1) \cdot \nu$; it is thus not necessary to explore the rest of the graph when a path of this length is found.

Next, we consider the time complexity of the algorithm on G_1. The algorithm explores all vertices except those on the last layer where it only opens some of them. On the last layer, it explores vertices adjacent to vertices with degrees k. The number of vertices on all layers except the last is $O(\Delta^{D-1})$. The number of vertices seen on the last layer is at most $\Delta \cdot \Delta^{D-2} = \Delta^{D-1}$. Overall, the complexity is thus $O(\Delta^{D-1})$. ∎

Open problems

It would be interesting to see whether the A* algorithm or some of its bidirectional variants [31] allow similarly strong guarantees as the bidirectional search.

Our proof of Theorem 1.1 works for multigraphs that allow parallel edges and self-loops. While we believe that the possibility of self-loops can be avoided, we are not sure whether the same holds for parallel edges: We believe that whether Theorem 1.1 holds in the setting of simple graphs is an interesting open question.

Acknowledgments

BH, RH, VR, and JT were partially funded by the Ministry of Education and Science of Bulgaria's support for INSAIT as part of the Bulgarian National Roadmap for Research Infrastructure. BH and RH were partially funded through the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (ERC grant agreement 949272). VR was partially funded through the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (ERC grant agreement 853109) RH and JT were supported by the VILLUM Foundation grant 54451. Part of this work was done while JT was working at and RH was visiting BARC at the University of Copenhagen. RH would like to thank Rasmus Pagh for hosting him there. RT's research at Princeton was partially supported by a gift from Microsoft. Part of this work was done during RT's visits to INSAIT and to the Simons Institute for the Theory of Computing.

References

[1] Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. "Instance-Optimal Geometric Algorithms". In: *J. ACM* 64.1 (Mar. 2017). ISSN: 0004-5411. DOI: 10.1145/3046673. URL: https://doi.org/10.1145/3046673.

[2] Noga Alon, Allan Grønlund, Søren Fuglede Jørgensen, and Kasper Green Larsen. "Sublinear time shortest path in expander graphs". In: *arXiv preprint arXiv:2307.06113* (2023).

[3] Ilya Baran and Erik D Demaine. "Optimal adaptive algorithms for finding the nearest and farthest point on a parametric black-box curve". In: *Proceedings of the twentieth annual symposium on Computational geometry*. 2004, pp. 220–229.

[4] Sabyasachi Basu, Nadia Kōshima, Talya Eden, Omri Ben-Eliezer, and C. Seshadhri. *A Sublinear Algorithm for Approximate Shortest Paths in Large Networks*. 2024. arXiv: 2406.08624 [cs.DS]. URL: https://arxiv.org/abs/2406.08624.

[5] Richard Bellman. "On a routing problem". In: *Quarterly of applied mathematics* 16.1 (1958), pp. 87–90.

[6] Thomas Bläsius, Cedric Freiberger, Tobias Friedrich, Maximilian Katzmann, Felix Montenegro-Retana, and Marianne Thieffry. "Efficient shortest paths in scale-free networks with underlying hyperbolic geometry". In: *ACM Transactions on Algorithms (TALG)* 18.2 (2022), pp. 1–32.

[7] Thomas Bläsius and Marcus Wilhelm. "Deterministic Performance Guarantees for Bidirectional BFS on Real-World Networks". In: *International Workshop on Combinatorial Algorithms*. Springer. 2023, pp. 99–110.

[8] Michele Borassi and Emanuele Natale. "KADABRA is an adaptive algorithm for betweenness via random approximation". In: *Journal of Experimental Algorithmics (JEA)* 24 (2019), pp. 1–35.

[9] Nairen Cao and Jeremy T Fineman. "Parallel exact shortest paths in almost linear work and square root depth". In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2023, pp. 4354–4372.

[10] Nairen Cao, Jeremy T Fineman, and Katina Russell. "Improved work span tradeoff for single source reachability and approximate shortest paths". In: 2020, pp. 511–513.

[11] Nairen Cao, Jeremy T. Fineman, and Katina Russell. "Efficient Construction of Directed Hopsets and Parallel Approximate Shortest Paths". In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2020. Chicago, IL, USA: Association for Computing Machinery, 2020, pp. 336–349. ISBN: 9781450369794. DOI: 10.1145/3357713.3384270. URL: https://doi.org/10.1145/3357713.3384270.

[12] Lijie Chen and Jian Li. "Open problem: Best arm identification: Almost instance-wise optimality and the gap entropy conjecture". In: *Conference on Learning Theory*. PMLR. 2016, pp. 1643–1646.

[13] Lijie Chen, Jian Li, and Mingda Qiao. "Towards instance optimal bounds for best arm identification". In: *Conference on Learning Theory*. PMLR. 2017, pp. 535–592.

[14] George B. Dantzig. *Linear Programming and Extensions*. Princeton: Princeton University Press, 1963. ISBN: 9781400884179. DOI: doi:10.1515/9781400884179. URL: https://doi.org/10.1515/9781400884179.

[15] Rina Dechter and Judea Pearl. "Generalized best-first search strategies and the optimality of A*". In: *Journal of the ACM (JACM)* 32.3 (1985), pp. 505–536.

[16] Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. "Adaptive set intersections, unions, and differences". In: *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*. 2000, pp. 743–752.

[17] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1 (1959), pp. 269–271.

[18] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Edsger Wybe Dijkstra: his life, work, and legacy*. 2022, pp. 287–290.

[19] Stuart E Dreyfus. "An appraisal of some shortest-path algorithms". In: *Operations research* 17.3 (1969), pp. 395–412.

[20] Ran Duan, Jiayi Mao, Xinkai Shu, and Longhui Yin. "A Randomized Algorithm for Single-Source Shortest Path on Undirected Real-Weighted Graphs". In: *arXiv preprint arXiv:2307.04139* (2023).

[21] Jürgen Eckerle, Jingwei Chen, Nathan Sturtevant, Sandra Zilles, and Robert Holte. "Sufficient conditions for node expansion in bidirectional heuristic search". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 27. 2017, pp. 79–87.

[22] Ronald Fagin, Amnon Lotem, and Moni Naor. "Optimal Aggregation Algorithms for Middleware". In: *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '01. Santa Barbara, California, USA: Association for Computing Machinery, 2001, pp. 102–113. ISBN: 1581133618. DOI: 10.1145/375551.375567. URL: https://doi.org/10.1145/375551.375567.

[23] Juan A Garay, Shay Kutten, and David Peleg. "A sublinear time distributed algorithm for minimum-weight spanning trees". In: *SIAM Journal on Computing* 27.1 (1998), pp. 302–316.

[24] Mohsen Ghaffari and Goran Zuzic. "Universally-optimal distributed exact min-cut". In: *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*. 2022, pp. 281–291.

[25] Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.

[26] Bernhard Haeupler, Richard Hladík, John Iacono, Václav Rozhoň, Robert Tarjan, and Jakub Tětek. *Fast and Simple Sorting Using Partial Information*. 2024. arXiv: 2404.04552 [cs.DS].

[27] Bernhard Haeupler, Richard Hladík, Václav Rozhoň, Robert Tarjan, and Jakub Tětek. *Universal Optimality of Dijkstra via Beyond-Worst-Case Heaps*. 2023. arXiv: 2311.11793 [cs.DS].

[28] Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. "Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality". In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 2022, pp. 1325–1338.

[29] Bernhard Haeupler, David Wajc, and Goran Zuzic. "Universally-optimal distributed algorithms for known topologies". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 2021, pp. 1166–1179.

[30] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[31] Robert C Holte, Ariel Felner, Guni Sharon, Nathan R Sturtevant, and Jingwei Chen. "MM: A bidirectional search algorithm that is guaranteed to meet in the middle". In: *Artificial Intelligence* 252 (2017), pp. 232–266.

[32] Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. *Simpler Optimal Sorting from a Directed Acyclic Graph*. 2024. arXiv: 2407.21591 [cs.DS].

[33] Ziyue Huang, Yuting Liang, and Ke Yi. "Instance-optimal mean estimation under differential privacy". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 25993–26004.

[34] Johannes Kirschner, Tor Lattimore, Claire Vernade, and Csaba Szepesvári. "Asymptotically optimal information-directed sampling". In: *Conference on Learning Theory*. PMLR. 2021, pp. 2777–2821.

[35] Tze Leung Lai and Herbert Robbins. "Asymptotically efficient adaptive allocation rules". In: *Advances in applied mathematics* 6.1 (1985), pp. 4–22.

[36] Zhaoqi Li, Lillian Ratliff, Kevin G Jamieson, Lalit Jain, et al. "Instance-optimal pac algorithms for contextual bandits". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 37590–37603.

[37] T Alastair J Nicholson. "Finding the shortest route between two points in a network". In: *The computer journal* 9.3 (1966), pp. 275–280.

[38] Ira Pohl. *Bi-directional and heuristic search in path problems*. Tech. rep. SLAC National Accelerator Laboratory (SLAC), Menlo Park, CA (United States), 1969.

[39] Tim Roughgarden. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2021. DOI: 10.1017/9781108637435.

[40] Václav Rozhoň, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. "Undirected (1+eps)-Shortest Paths via Minor-Aggregates: Near-Optimal Deterministic Parallel and Distributed Algorithms". In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2022. Rome, Italy: Association for Computing Machinery, 2022, pp. 478–487. ISBN: 9781450392648. DOI: 10.1145/3519935.3520074. URL: https://doi.org/10.1145/3519935.3520074.

[41] Václav Rozhoň, Bernhard Haeupler, Anders Martinsson, Christoph Grunau, and Goran Zuzic. "Parallel breadth-first search and exact shortest paths and stronger notions for approximate distances". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*. 2023, pp. 321–334.

[42] A Schrijver. "On the History of Combinatorial Optimization (till 1960)". In: *Handbook of Discrete Optimization/Elsevier* (2005).

[43] Eshed Shaham, Ariel Felner, Nathan R Sturtevant, and Jeffrey S Rosenschein. "Optimally Efficient Bidirectional Search." In: *IJCAI*. 2019, pp. 6221–6225.

[44] Nathan Sturtevant and Ariel Felner. "A brief history and recent achievements in bidirectional search". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[45] Gregory Valiant and Paul Valiant. "An automatic inequality prover and instance optimal identity testing". In: *SIAM Journal on Computing* 46.1 (2017), pp. 429–455.

[46] Gregory Valiant and Paul Valiant. "Instance optimal learning of discrete distributions". In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. 2016, pp. 142–155.

[47] Goran Zuzic, Goramoz Goranci, Mingquan Ye, Bernhard Haeupler, and Xiaorui Sun. "Universally-Optimal Distributed Shortest Paths and Transshipment via Graph-Based L1-Oblivious Routing". In: *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2022.