Fast Robust Shortest Path Computations

Christoph Hansknecht
Institute for Mathematical Optimization, Technical University Braunschweig, Germany
c.hansknecht@tu-braunschweig.de

Alexander Richter
Institute for Mathematical Optimization, Technical University Braunschweig, Germany
a.richter@tu-braunschweig.de

Sebastian Stiller
Institute for Mathematical Optimization, Technical University Braunschweig, Germany
sebastian.stiller@tu-braunschweig.de

Abstract

We develop a fast method to compute an optimal robust shortest path in large networks like road networks, a fundamental problem in traffic and logistics under uncertainty.

In the robust shortest path problem we are given an s-t-graph $D(V, A)$ and for each arc a nominal length $c(a)$ and a maximal increase $d(a)$ of its length. We consider all scenarios in which for the increased lengths $c(a) + \bar{d}(a)$ we have $\bar{d}(a) \leq d(a)$ and $\sum_{a \in A} \frac{\bar{d}(a)}{d(a)} \leq \Gamma$. Each path is measured by the length in its worst-case scenario. A classic result [6] minimizes this path length by solving $(|A|+1)$-many shortest path problems. Easily, $(|A|+1)$ can be replaced by $|\Theta|$, where Θ is the set of all different values $d(a)$ and 0. Still, the approach remains impractical for large graphs.

Using the monotonicity of a part of the objective we devise a Divide and Conquer method to evaluate significantly fewer values of Θ. This methods generalizes to binary linear robust problems. Specifically for shortest paths we derive a lower bound to speed-up the Divide and Conquer of Θ. The bound is based on carefully using previous shortest path computations. We combine the approach with non-preprocessing based acceleration techniques for Dijkstra adapted to the robust case.

In a computational study we document the value of different accelerations tried in the algorithm engineering process. We also give an approximation scheme for the robust shortest path problem which computes a $(1+\epsilon)$-approximate solution requiring $\mathcal{O}(\log(\hat{d}/(1+\epsilon)))$ computations of the nominal problem where $\hat{d} := \max d(A)/\min(d(A) \setminus \{0\})$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Graph Algorithms, Shortest Paths, Robust Optimization

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.5

1 Introduction

We develop an algorithm for the cost-robust shortest path problem that significantly reduces the time needed to compute such paths on road networks in practice.

Finding a shortest path from a source s to a sink t in a graph with arc lengths $c(a)$ is a basic algorithmic problem with numerous applications, prominently involving navigation in road networks. Dijkstra's algorithm is the backbone of most navigation applications, but it requires modern acceleration techniques to find within fractions of seconds a route in a network with several hundred thousands or millions of arcs, e.g., in the European road network.

18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018).
Editors: Ralf Borndörfer and Sabine Storandt; Article No. 5; pp. 5:1–5:21

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Unfortunately, input data in real-world applications is usually subject to changes, uncertainty or error. For travel times on roads, i.e., arc lengths in shortest path calculations, the change of data is often caused by varying traffic. Several approaches have been proposed to address this problem, including prediction of traffic, leading to time dependent travel times, as well as stochastic models. In this paper we study the classical cost-robust shortest path problem introduced by Bertsimas and Sim. Cost-robust optimization is an alternative approach to handle varying and uncertain data. It minimizes the cost a solution attains in its specific worst-case scenario out of a given set of scenarios. The advantage of the robust approach is that – within the limits of the scenario set – the objective is a deterministic, guaranteed upper bound on the actual travel time.

The scenario set for cost-robustness introduced by Bertsimas and Sim allows each cost coefficient $c(a)$ of a linear cost function to deviate up to a – individual for each variable x_a – maximal deviation $d(a)$. In addition, the number of deviations in a scenario is limited by an input parameter Γ. This is equivalent to limiting by Γ the sum of the fractions of maximal deviations occurring in a scenario. Formally, for a given set of binary variables $\{x_a, a \in A\}$ and vectors c and d in $\mathbb{N}^{|A|}$ the scenario set for the cost-functions is:

$$\left\{ c + \bar{d} : 0 \leq \bar{d}(a) \leq d(a), \forall a \in A \wedge \sum_{a \in A} \frac{\bar{d}(a)}{d(a)} \leq \Gamma \right\}. \tag{1}$$

For this scenario set the cost-robust counterpart of any binary linear program can be solved by solving at most $(|A| + 1)$–many identical binary linear programs with different linear cost functions. More precisely, let Θ contain 0 and all $d(a)$. Then one has to solve the problem for each $\theta \in \Theta$ and the cost function $\Gamma\theta + \sum_A x_a(c(a) + \max(d(a) - \theta, 0))$. Intuitively, the θ enumerates over the smallest deviation $d(a)$ occurring in the scenario. This highly cited result by Bertsimas and Sim applies to cost-robust shortest path, which can thus be found by solving one standard shortest path problem for each arc in the graph.

For a road network with several hundred thousand or millions of arcs this is impractical even when using fast shortest path algorithms. Therefore, we devise a method to significantly reduce the computational effort.

Starting from the Bertsimas and Sim result we use three ways towards practically useful cost-robust shortest path methods. First we reduce the number of θ-values to be examined. Second, we use fast shortest path methods. Third, we reuse previous computations for bounds and goal-directed search, further accelerating the shortest path computations.

It has been proposed [21] that a cost-robust binary problem can be solved by Γ-many copies of the nominal problem. Unfortunately, this result contains a subtle error. We give a counter-example in the appendix which hints to our conviction that essentially $|\Theta|$-many shortest path computations are needed in general.

Accelerated shortest path methods differ on whether they use preprocessing of the graph or not. In this paper, we restrict ourselves to not preprocess the graph. We instead use goal-directed and bidirectional search and adapt both to the cost-robust setting. The high deviations in the arc length in the robust case inhibit the use of traditional preprocessing techniques used for deterministic shortest paths.

1.1 Our contribution

- We give an approximation scheme for general robust combinatorial optimization problems which can be used to compute a $(1 + \epsilon)$-approximate solution using $\mathcal{O}(\log(\hat{d}/(1 + \epsilon)))$ computations of the original problem.

- We introduce a Divide and Conquer approach together with lower bounds for general robust combinatorial optimization problems which can be used to reduce the number of computations of the original problem. The reduction of computations is achieved by carefully reducing the number of θ-values to be considered.
- When applying this to the robust shortest path problem we additionally accelerate the computations of individual shortest paths using pruning and a goal-directed search tailored to the robust shortest path problem.
- We give an efficient method to obtain lower bounds for the length of shortest paths with respect to c_θ. We use these bounds to speed up the Divide and Conquer approach.
- We conduct a computational study showing the effectiveness of our techniques.

1.2 Organization of this paper

We begin by formally introducing the robust shortest path problem in Section 2. We restate the main theorem by Bertsimas and Sim and devise an approximation scheme for the robust shortest path problem. In Section 3 we propose a general framework designed to reduce the number of computations of shortest path computations required to solve a robust shortest path problem. The framework relies on Theorem 3 which is based on the fact that the costs of arcs are non-increasing with respect to θ. We augment this framework by applying shortest path acceleration techniques to the robust shortest path problem. These techniques are search pruning (see Section 4) and goal-direction (see Section 5). The Divide and Conquer framework relies on lower bounds in order to remove dominated values. In Section 6 we devise a method to derive lower bounds of high quality based on information obtained from previous shortest path computations. We include these lower bounds into our Divide and Conquer approach. In order to show the effectiveness of our approach we conduct a computational experiment in Section 7.

1.3 Related work

Robust optimization evolved as a vivid research field during the past decade and shows a broad range of applications, for recent surveys we refer to [5] and [13]. The popularity of robust optimization is in part due to a large area of applications such as network design and routing problems. Network design problem in particular suffer from uncertainty with respect to demands and construction costs. These uncertainties can be treated by adding robustness to the underlying model [3, 20]. Robustness against demand uncertainty is also an important topic in problems such as vehicle routing [11] and lot sizing [22].

An important question with respect to robust optimization is whether or not tractability is preserved for the robust counterparts of polynomially solvable problems. Whether or not this is the case depends on properties of the nominal problem as well as on the employed robust model. For some choices of models, such as minmax regret models, nominally tractable problems become NP-hard (see for example [12]). In contrast, in [6] Bertsimas and Sim introduced a very general robust model which can be applied to many combinatorial optimization problems while preserving tractability.

The model of Bertsimas and Sim has gained wide acceptance and formed a basis for the study of robust combinatorial optimization problems, in particular regarding problems related to the robustness of shortest paths. Büsing considered the problem of robustness and robust recoverability in [8, 7]. In this setting, after a robust scenario has been realized it is still possible to perform some modifications of the previously chosen path in order to recover from the incurring robust costs. The authors of [19] considered the robust shortest path problem

with respect to robust costs corresponding to a product of two factors attained according to the model of Bertsimas and Sim. In [21], Poss considered combinatorial problems which can be solved with a dynamic programming approach. The author claimed that the robust counterparts of such problems can be solved with a dynamic program with a size increased by at most Γ. Unfortunately the proof contains a subtle error and the result does not hold. We give a counter-example in the appendix.

Since the ordinary shortest path problem has many real-world applications, considerable effort was put into an accelerated computation. Over the years, different preprocessing techniques such as arc flags [18] and contraction hierarchies [14] were introduced (see [4] for a summary). Preprocessing techniques require an initial offline phase which is used to augment the underlying problem in order to speed up queries in a subsequent online phase. The techniques perform very well in practice, decreasing query times by several orders of magnitude. It was shown in [1] that the query time with respect to preprocessing techniques decreases asymptotically for graphs with low *highway dimension*, a requirement generally satisfied for road networks. A related area of research considers large-scale networks which occur for example in social graphs. Such networks can comprise more than a billion vertices some of which having extremely large degrees. Conventional preprocessing techniques can't be applied in this case. The authors of [9, 16] introduced an inexact preprocessing based on landmarks which is comparable to the approach in [15] for road networks. In contrast the authors of [2] considered a preprocessing technique which either answers the query correctly (in more than 99 % of the queries conducted in their experiments) and fails otherwise.

2 The robust shortest path problem

The robust shortest path problem is defined on a directed graph $D = (V, A)$ with n vertices and m arcs. Each arc $a \in A$ has costs $c(a) \in \mathbb{N}$ and deviations $d(a) \in \mathbb{N}$. A parameter $\Gamma \in \mathbb{N}$ governs the conservatism in accordance with the model of Bertsimas and Sim. Specifically, consider a path P given as a sequence of arcs. A worst-case scenario in the scenario set defined by (1) can be assumed to increase the costs on Γ of the arcs belonging to P to the upper bound d, yielding a total cost of

$$\sum_{a \in P} c(a) + \max_{\substack{S \subseteq P \\ |S| \leq \Gamma}} \sum_{a \in S} d(a). \tag{2}$$

The following theorem shows that the robust shortest path problem can be solved in polynomial time. This theorem and its proof will form the basis of this paper.

▶ **Theorem 1** (Bertsimas and Sim in [6]). *The robust shortest path problem can be solved using at most $m + 1$ computations of nominal shortest paths.*

Proof. We are attempting to find a path minimizing the cost given by (2). We first consider a fixed path P and rewrite the inner optimization problem in terms of variables denoting membership in the set S:

$$\begin{aligned} \max \quad & \sum_{a \in P} x(a) \cdot d(a) \\ \text{s.t.} \quad & \sum_{a \in P} x(a) \leq \Gamma \\ & 0 \leq x(a) \leq 1 \quad \forall a \in P \end{aligned} \tag{3}$$

This program has the following dual:

$$\begin{aligned} \min\ & \Gamma\theta + \sum_{a\in P} y(a) \\ \text{s.t. } & y(a) + \theta \geq d(a) \quad \forall a \in P \\ & \theta, y(a) \geq 0 \qquad\ \ \forall a \in P \end{aligned} \tag{4}$$

It is easy to see that $y(a)$ can be fixed to $\max(d(a) - \theta, 0)$. As a result, minimizing (2) is equivalent to finding a path P minimizing

$$\min_{\theta\in\mathbb{R}_{\geq 0}} \Gamma\theta + \sum_{a\in P} c(a) + \max(d(a) - \theta, 0) \tag{5}$$

The function $\theta \mapsto \max(d(a) - \theta, 0)$ is piecewise linear with a break point at $d(a)$. Therefore the function

$$\theta \mapsto \min_{P\in\mathcal{P}} \Gamma\theta + \sum_{a\in P} c(a) + \max(d(a) - \theta, 0) \tag{6}$$

has break points at $d(a)$ for each $a \in A$. It will therefore attain its minimum either at 0 or at some $d(a)$. Thus, a robust shortest path can be found with at most $m + 1$ nominal shortest path computations according to the costs defined by the corresponding values of θ. ◀

Even though the shortest path problem is easily solvable in practice, the overhead of solving $m + 1$ variants renders the robust counterpart intractable in practice. Observe that the number of shortest path computations required in total does not actually depend on the number of arcs but rather on the cardinality of the set

$$\Theta := \{0\} \cup \{d(a) \mid a \in A\}. \tag{7}$$

This suggests an approximation scheme based on solving an instance with a lower number of deviations:

▶ **Theorem 2.** *Let $\hat{d} := \max d(A)/\min(d(A) \setminus \{0\})$, $\epsilon > 0$. A $(1 + \epsilon)$-approximate solution of the robust shortest path problem can be computed with $\mathcal{O}(\log(\hat{d}/(1 + \epsilon)))$ computations of the nominal shortest path problem.*

Proof. Let $\bar{d} : M \mapsto \mathbb{R}_{\geq 0}$ be the values of d rounded up to the next power of $(1 + \epsilon)$:

$$\bar{d}(a) := (1 + \epsilon)^{\lceil \log_{1+\epsilon}(d(a)) \rceil} \quad \forall\, a \in A. \tag{8}$$

There are only $\mathcal{O}(\log(\hat{d}/(1 + \epsilon)))$ different values for θ with respect to $\bar{d}$, which implies that we have to solve only that many instances of the original problem in order to obtain a robust optimum with respect to $\bar{d}$. Let P be a solution of the robust problem with respect to the deviations d. Let $S \subseteq P$ be the set of at most Γ entries causing the robust cost contribution to P with respect to d. In the worst case, every $d(a)$ increases by a factor of less than $(1 + \epsilon)$ from d to $\overline{d}$. Thus, the robust cost contribution with respect to $\overline{d}$ is again caused by the entries in S, increasing the cost of P by less than $(1 + \epsilon)$. ◀

▶ Remark.

1. The approximation guarantee is tight: Consider an instance of the robust shortest path problem given by a digraph consisting of two parallel arcs with pairs of costs and deviations of $(\epsilon/2, (1+\epsilon)^k + \epsilon/2)$ and $(0, (1+\epsilon)^{k+1})$, a parameter of $k \in \mathbb{N}_{>0}$ and $\Gamma = 1$. The robust shortest path has a cost of $\epsilon + (1 + \epsilon)^k$, whereas a robust shortest path for the rounded instance costs $(1 + \epsilon)^{k+1}$ in the original instance. As $k \to \infty$ a ratio of $1 + \epsilon$ is achieved.

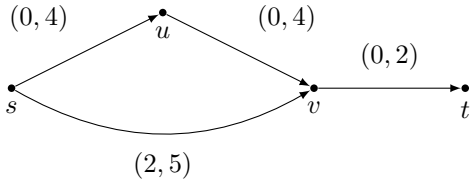


Figure 1 An example for robust shortest paths not forming a tree. Pairs of numbers on arcs represent costs and deviations.

2. Bertsimas and Sim show that robust minimum cost network flow problems can be approximated to a factor of $(1+\epsilon)$ in $\mathcal{O}(\log(m\bar{\theta}/\epsilon))$, where $\bar{\theta} \coloneqq \max_{a\in A} u_a d_a$ for capacities u. However, robust network flows are not generally integral for integral capacities. Specifically, a robust network flow of one unit no longer corresponds to a path.
3. Recall that the shortest paths problem exhibits an optimal substructure: All shortest paths leaving a common source vertex s can be chosen to form a tree in the underlying graph. This does no longer hold for robust shortest paths, as shown in Figure 1: For $\Gamma = 2$ the unique robust shortest path from s to t leads past vertex u, causing a cost of 8. The robust shortest (s, v) path consists solely of the lower arc.

3 Divide and Conquer

In this section we will describe the main idea used to reduce the number of θ-values which have to be considered to compute a robust shortest path based on Theorem 1. We define $c_\theta(a) \coloneqq c(a) + \max(d(a) - \theta, 0)$ and observe that this term is non-increasing in θ. The same holds for the cost of a path P defined as $c_\theta(P) \coloneqq \sum_{a\in P} c_\theta(a)$. For a fixed θ we let

$$c^{\text{opt}}(\theta) \coloneqq \min_{P\in\mathcal{P}(s,t)} c_\theta(P). \tag{9}$$

Since $c^{\text{opt}}(\theta)$ is the minimum of non-increasing functions, it is non-increasing as well. In order to find a robust shortest path we will minimize the function

$$C^\Gamma(\theta) \coloneqq \Gamma\theta + c^{\text{opt}}(\theta). \tag{10}$$

If $C^\Gamma(\theta)$ were a convex function in θ, we could use binary search or similar techniques in order to reduce the number of required shortest path computations. Unfortunately $C^\Gamma(\theta)$ is not generally convex. We can however derive the following theorem from the fact that $c^{\text{opt}}(\theta)$ is non-increasing:

▶ **Theorem 3.** *Let $\theta_{\min} < \theta_{\max}$ be in Θ and $\theta \in \Theta \cap (\theta_{\min}, \theta_{\max})$.*
1. *If $c^{\text{opt}}(\theta_{\min}) = c^{\text{opt}}(\theta_{\max})$, then it holds that $C^\Gamma(\theta) \geq C^\Gamma(\theta_{\max})$.*
2. *Let θ^* be in Θ. If $\Gamma\theta + c^{\text{opt}}(\theta_{\max}) \geq C^\Gamma_{\theta^*}$, then the minimum over C^Γ is not attained in $[\theta, \theta_{\max})$.*

Proof. For the first part note that since c^{opt} is non-increasing we have that $c^{\text{opt}}(\theta) = c^{\text{opt}}(\theta_{\min}) = c^{\text{opt}}(\theta_{\max})$. The result then follows from the definition of C^Γ. Turning to the second part, we let $\theta' \in [\theta, \theta_{\max})$. We know that $C^\Gamma(\theta') \geq \Gamma\theta + c^{\text{opt}}(\theta_{\max}) \geq C^\Gamma(\theta^*)$ and therefore $C^\Gamma(\theta)$ is at least $C^\Gamma(\theta^*)$. ◀

Both cases of Theorem 3 enable us to discard an interval of possible values for θ. We therefore use a Divide and Conquer approach as a general framework to speed up computations. The approach works as follows: We maintain a set of intervals of values in Θ together with

Algorithm 1: A Divide and Conquer algorithm for the robust shortest path problem.

Algorithm DivideAndConquer
- **Input:** Digraph D, costs c, deviations d, parameter Γ, vertices s, t
- **Output:** A robust shortest (s,t)-path
- $S \leftarrow \{\Theta\}$
- $\theta^* \leftarrow$ The value of $\min(\Theta)$, $\max(\Theta)$ with lower C^Γ
- **while** $S \neq \emptyset$ **do**
  - $I_{\min} \leftarrow$ The interval I from S with the lowest $\min(C^\Gamma(\min(I)), C^\Gamma(\max(I)))$
  - **if** $I_{\min}$ can be discarded **then**
    - **continue**
  - $I_{\min} \leftarrow$ Remove dominated values from $I_{\min}$
  - $(I_{\text{low}}, I_{\text{high}}) \leftarrow$ Intervals such that $I_{\text{low}} \cup I_{\text{high}} = I_{\min}$, $|I_{\text{low}} \cap I_{\text{high}}| = 1$, and $||I_{\text{low}}| - |I_{\text{high}}|| \leq 1$
  - $\theta_{\text{median}} \leftarrow$ The median value, single element in $I_{\text{low}} \cap I_{\text{high}}$
  - $\theta^* \leftarrow$ The value of θ^*, θ_{median} with lower C^Γ
  - $S \leftarrow S \cup \{I_{\text{low}}, I_{\text{high}}\}$
- **return** The path corresponding to θ^*

the currently best (w.r.t. C^Γ) known value θ^*. We also ensure that the shortest paths with respect to the minimum / maximum of each interval are computed before the interval is considered. At each step of the algorithm we select the interval which has the lowest value of C^Γ at an endpoint. We first use Theorem 3 to try to discard the interval. If the interval can't be discarded we proceed to remove any dominated values. We split the resulting interval into two halves which share exactly one value in Θ, compute the shortest path with respect to that value and decide whether or not to replace θ^*. We then add the two intervals to the set and continue. The details are outlined in Algorithm 1.

Note also that Theorems 3 and 2 (and therefore also Algorithm 1) work for arbitrary robust combinatorial optimization problems.

4 Search pruning

Dijkstra's algorithm explores a graph by *labeling* and *settling* vertices. A vertex is labeled when it is first explored. As soon as a shortest path connecting the vertex is known, the vertex is declared to be settled. Since we compute shortest (s,t)-paths for multiple cost functions c_θ, we reuse information we have gathered from previous computations in order to decrease the number of vertices which have to be labeled / settled in subsequent iterations of Dijkstra's algorithm. The following theorem gives a sufficient condition for excluding vertices during searches:

▶ **Theorem 4.** *Let v be a vertex and $\theta < \theta'$ where $\theta, \theta' \in \Theta$. Let $P_\theta, P_{\theta'}$ be (s,v)-paths that are optimal with respect to c_θ respectively $c_{\theta'}$. Let*

$$\Gamma\theta + c_\theta(P_\theta) > \Gamma\theta' + c_{\theta'}(P_{\theta'}). \tag{11}$$

Then a robust shortest (s,t)-path is either attained for a value $\neq \theta$ or it does not contain v.

Proof. The proof may be found in Appendix A. ◀

We can make the most of this theorem when we evaluate the values of θ in a decreasing fashion. During these computations we maintain a map $\bar{C} : V \to \mathbb{R}_{\geq 0}$. Think of $\bar{C}(v)$ as a known upper bound on the cost of a robust (s, v)-path which we initialize to $\bar{C} \equiv \infty$. When we settle a vertex $u \neq t$ during the search for a shortest path with respect to c_θ we investigate each outgoing arc $(u, v) \in A$. The path leading to u together with (u, v) forms a path P leading to v yielding a value $\Gamma\theta + c_\theta(P)$. If $\Gamma\theta + c_\theta(P) > \bar{C}(v)$ we do not have to label v. Otherwise we label v and decrease $\bar{C}(v)$ to $\Gamma\theta + c_\theta(P)$.

5 Goal-direction

A common extension of Dijkstra's algorithm is known as goal-directed search, introduced in [17]. It is based on a potential $\pi : V \to \mathbb{R}_{\geq 0}$ such that the corresponding reduced costs $c^\pi(u, v) := c(u, v) - \pi(u) + \pi(v)$ are non-negative for each $(u, v) \in A$. It is possible to derive a potential while searching for a shortest path. Consider a search from t in the direction of s. The resulting (partial) shortest-path tree $T = (V(T), A(T))$ is rooted at t and contains all settled vertices. For each $v \in V(T)$ we obtain a path $P(v, t)$ leading from v to the t. Let $c_{\max}(T)$ be the maximum value of $c(P(v, t))$ for $v \in V(T)$. It is then easy to see that the following function is a potential:

$$\pi(v) := \begin{cases} c(P(v,t)) & \text{for } v \in V(T) \\ c_{\max}(T) & \text{otherwise.} \end{cases} \tag{12}$$

In the robust setting, a potential with respect to c_θ is also a potential for $c_{\theta'}$ with $\theta' < \theta$ (since $c_{\theta'} \geq c_\theta$). We use this observation in the following way: We first compute the potential (12) with respect to $\theta_{\max}$ while finding the corresponding path using a backward search. In subsequent forward searches with respect to smaller values in Θ we use this potential. If the costs with respect to θ and $\theta_{\max}$ coincide, the arcs in the backward tree will have zero reduced cost. If all other arcs have nonzero reduced cost, then only the arcs in the shortest paths will have to be settled, greatly decreasing computation time. Intuitively, if θ and $\theta_{\max}$ are close, then the potential computed from $\theta_{\max}$ is an excellent choice for the search with respect to θ.

6 Divide and Conquer for robust shortest paths

We refine Algorithm 1 by exploiting structural properties of the robust shortest path problem. We present our results for a unidirectional search here. In the appendix we show an extension to goal-directed and bidirectional searches in a more general setting.

Consider some interval $I := [\theta_{\min}, \theta_{\max}]$ which appears in the course of Algorithm 1. As an invariant we have completed the Dijkstra search for $\theta_{\min}$. We want to reuse labeling information of this search to derive lower bounds on $C^\Gamma_{\theta_0}$ for some $\theta_0 \in I$. If such a lower bound exceeds the best known upper bound for C^*, we disregard θ_0. In order to accelerate the computation of a robust shortest path, the computation of the lower bound for $C^\Gamma_{\theta_0}$ must be significantly faster than a computation of the path for c_{θ_0}.

We argue about a hypothetical (s, t)-path P and its cost $c_\theta(P)$. The cost is non-increasing and piecewise linear as a function in θ. It has breakpoints whenever θ increases beyond $d(a)$ for some $a \in P$. From this point on the cost $c_a(\theta)$ stays constant at $c(a)$. We know the values $c^{\text{opt}}(\theta_{\min})$ and $c^{\text{opt}}(\theta')$ for some values $\theta' \geq \theta_{\max}$. Whatever the value of $c_{\theta_0}(P)$, the cost of P cannot decrease below these amounts when evaluated at the respective values (see Figure 2).

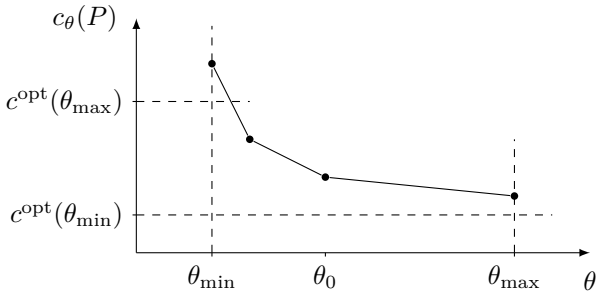


Figure 2 The cost $c_\theta(P)$ of some P. The cost at θ_0 has to be consistent with $c^{\text{opt}}(\theta_{\min})$, $c^{\text{opt}}(\theta_{\max})$.

We go on to formulate a mixed integer program (shown in (13)) to choose an arc set minimizing c_{θ_0}. To make the formulation as strong as possible we choose the smallest possible set of arcs to include into this program: Let $M \subset A$ be the set of scanned arcs, i.e. arcs having a tail which has been settled throughout the search for the shortest (s,t)-path for $\theta_{\min}$. Furthermore, let $M_{\theta_{\min}} \subseteq M$ be the restriction of M to *active* arcs i.e. arcs with $d(a) > \theta_{\min}$. It turns out to be sufficient to consider the arcs in $M_{\theta_{\min}}$ to obtain a lower bound on c_{θ_0}.

We introduce a binary variable x_a for each $a \in A$ denoting whether or not a is contained in P. The variable y models a lower bound on the cost $c_{\theta_{\min}}(P)$ of P yielding (13b). The negative slope of $c_\theta(P)$ at the point $\theta_{\min}$ corresponds to the number of active arcs in P. In the worst case we have $c_\theta(P) = y - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta) - \theta_{\min})$ by subtracting from y the contribution of the active arcs. In this case the objective (13a) equals $c_{\theta'}(P)$. However, not all active arcs from $M_{\theta_{\min}}$ can occur in P because for such a path P the value of $c_{\theta'}(P)$ might violate our observations of shortest path lengths for $c^{\text{opt}}(\theta')$. Thus we must raise the variable y to have $c_{\theta'}(P) \geq c^{\text{opt}}(\theta')$. Using the expression for $c_\theta(P)$ from above we obtain (13c) and altogether the following theorem:

▶ **Theorem 5.** *Given an arc set M of scanned arcs during a completed unidirectional search for $c_{\theta_{\min}}$, then a lower bound $O_{\theta_0} \leq c^{opt}(\theta_0)$ is given by*

$$
\begin{aligned}
O_{\theta_0} = \min\ & y - \sum_{a \in M_{\theta_{\min}}} x_a \cdot (\min(d(a), \theta_0) - \theta_{\min}) && \text{(13a)}\\
\text{s.t.}\ & y \geq c^{opt}(\theta_{\min}) && \text{(13b)}\\
& y - \sum_{a \in M_{\theta_{\min}}} x_a \cdot (\min(d(a), \theta') - \theta_{\min}) \geq c^{opt}(\theta') && \\
& \quad \forall\, \theta' > \theta_0 \text{ with known } c^{opt}(\theta') && \text{(13c)}\\
& y \geq 0,\ x \in \{0,1\}^{M_{\theta_{\min}}} && \text{(13d)}
\end{aligned}
$$

The theorem can in fact be further generalized to the bidirectional, goal-directed case. The generalized Theorem 9 and its proof may be found in Appendix A.

The following theorem states that bounds O_θ obtained for multiple θ by Theorem 5 are nonincreasing in θ. This observation can reduce the number of necessary bound computations throughout our algorithm. A proof follows from the more general Theorem 10 in Appendix A.

▶ **Theorem 6.** *For each $\theta_{\min} < \theta_0 < \theta_1$ we have $c^{opt}(\theta_{\min}) \geq O_{\theta_0} \geq O_{\theta_1} \geq c^{opt}(\theta')$ for all θ' that were considered in* (13c) *for both O_{θ_0} and O_{θ_1}.*

It is too time-consuming to solve (13) in order to compute a single bound. We therefore consider a relaxation of the program which can be solved a lot faster while still providing sufficient bounds. Observe that the program has the structure of a multi-dimensional

knapsack problem once we fix some value of y. We first relax the integrality of x towards $x \in [0,1]^{M_{\theta_{\min}}}$ and consider a single value $\theta' = \theta_{\max}$ for (13c). What remains is a fractional one-dimensional knapsack problem where arcs correspond to knapsack items:

$$\begin{aligned} \max \quad & \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_0) - \theta_{\min}) - y \\ \text{s.t.} \quad & c^{\text{opt}}(\theta_{\min}) \leq y \\ & \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_{\max}) - \theta_{\min}) \leq y - c^{\text{opt}}(\theta_{\max}) \\ & x \in [0,1]^{M_{\theta_{\min}}} \end{aligned} \tag{14}$$

Suppose we fix $y = c^{\text{opt}}(\theta_{\min})$. The optimum of the relaxation can be obtained by selecting items greedily w.r.t. their gain, i.e. $\text{gain}(a) \coloneqq (\min(d(a), \theta_0) - \theta_{\min})/(\min(d(a), \theta_{\max}) - \theta_{\min})$. This leaves exactly one split item a with fractional value for x_a. We argue that increasing y further is not beneficial: An increase of y by ϵ will increase the capacity of the knapsack by ϵ and thereby lead to increased x_a in a greedy optimum. The objective function changes by $\epsilon(\text{gain}(a) - 1)$ which is nonpositive because $\text{gain}(a) \leq 1$ for all arcs. It is therefore never advisable to increase y and we only have to sort the arcs in $M_{\theta_{\min}}$ w.r.t. their gain in order solve problem (14) and obtain a bound O_{θ_0}. Observe that

$$\text{gain}(a) = \begin{cases} (\theta_0 - \theta_{\min})/(\theta_{\max} - \theta_{\min}) & \text{if } d(a) \geq \theta_{\max} \\ (\theta_0 - \theta_{\min})/(d(a) - \theta_{\min}) & \text{if } d(a) < \theta_{\max} \text{ and } d(a) \geq \theta_0 \\ (d(a) - \theta_{\min})/(d(a) - \theta_{\min}) = 1 & \text{if } d(a) < \theta_{\max} \text{ and } d(a) < \theta_0 \end{cases} \tag{15}$$

Thus the value $\text{gain}(a)$ decreases as $d(a)$ increases and it is sufficient to sort the arcs in $M_{\theta_{\min}}$ once according to $d(a)$ in order to compute O_{θ_0} for each $\theta_0 \in \Theta \cap (\theta_{\min}, \theta_{\max})$. We incorporate this *relaxed knapsack bound (RKB)* into the Divide and Conquer approach and apply the generalization of Theorem 5 to goal-directed and bidirectional search.

▶ Remark (Preprocessing). As mentioned above, preprocessing techniques for the ordinary shortest path problem have been extensively studied in the past. Specifically, successful attempts have been made [10] to adapt preprocessing techniques to problems with time-dependent cost functions. Therefore it seems obvious to investigate these techniques with respect to applicability to the robust shortest path problem.

Existing preprocessing techniques operating on problems with changing cost functions generally rely on the ability to provide reasonable bounds on the values attained by the cost functions in order to prune the search space efficiently. Unfortunately, the costs of arcs vary widely between c and $c + d$ in the robust shortest path problem, making it impossible to derive meaningful bounds. As a result we were not able to find preprocessing techniques leading to a significant decrease in query time.

7 Computational experiments

7.1 Experimental network

Due to the long history of experimental evaluations of shortest path algorithms, instances of road networks are ready at hand. However, these networks generally lack data necessary to determine deviation values. Furthermore, shortest path experimentation is conducted on continent-sized networks which are as of yet too large to allow for the computation of robust shortest paths.

We therefore chose to construct a road network ourselves. To this end, we considered a subnetwork of the German road network given by the region of Lower Saxony[1]. We performed the following preprocessing steps in order to obtain a network suitable for routing purposes:

1. We filtered the file to only include ways with `highway` tags, excluding certain highway types such as tracks / service road etc. This process yielded 1.93M nodes and 0.36M ways.
2. We constructed a graph by replacing ways with sequences of arcs, adjusting for one-way restrictions. The resulting graph has 1.93M vertices and 2.17M arcs.
3. We removed directed and undirected chains from the graph. Chains occur frequently as they are used to model the curvature of roads. Therefore the resulting graph shrinks to 0.37M vertices and 0.50M arcs.
4. Since queries for robust paths in an insufficiently connected graph skew computational results we extracted the largest (in terms of the number of vertices) strongly connected component which has 0.15M vertices and 0.23M arcs.

We defined the values of c and d on the network as follows: The nominal length c is defined as the time needed to traverse a segment in accordance with the legal speed limit. To define d we assumed that a certain number of segments is affected by situations such as traffic accidents or road works. If a segment a is affected, the traveling speed drops from the legal speed limit to a value of at most 10 km/h. The value d is chosen such that $c + d$ corresponds to the travel time according to a speed of at most 10 km/h (where $d(a) = 0$ if the speed limit of a is already at most 10 km/h). To avoid numerical problems we rounded both c and d to the nearest second, resulting in $|\Theta| = 1,043$ different deviation values[2]. We further assumed that at most $\Gamma = 5$ road segments suffer from additional congestion.

7.2 Experimental methodology

In order to judge the performance of a shortest path algorithm, the query time of the algorithm is compared to that of Dijkstra's algorithm without any preprocessing applied. This approach raises the following issue: The time to answer a query for a shortest (s, t)-path using Dijkstra's algorithm is highly dependent on the choice of the vertices s and t: If the distance of s and t w.r.t. c is small compared to the diameter of D, then the search explores only a small part of D and finishes quickly. If on the other hand s and t are far apart, then almost the entire graph is explored before a path is found.

This issue can be addressed with the notion of a Dijkstra rank: A search from a fixed source s using Dijkstra's algorithm will settle the vertices in D in the order[3] $s = v_1, v_2, \ldots, v_k$. We define the *Dijkstra rank* of v_j with respect to s as the value j. Note that the distance from s to v_j is non-decreasing and the query time using Dijkstra's algorithm is increasing in the Dijkstra rank. For a pair (s, t) of vertices we define the Dijkstra rank of (s, t) by the Dijkstra rank of t with respect to s.

In order to evaluate the performance of different robust shortest path algorithms we recorded the query time for randomly chosen pairs of vertices with similar Dijkstra ranks. More specifically, we selected pairs of vertices with ranks in $[l \cdot n, u \cdot n)$ where l and u form intervals of size of 10 % of $|V|$.

[1] The initial data was obtained from the OpenStreetMap project, see `https://www.openstreetmap.org`.
[2] The accompanying data may be found at 10.6084/m9.figshare.c.4193588.
[3] We assume that ties are broken consistently.

For each interval we measured the average query time for a sample of 500 random pairs of vertices in order to reduce measurement errors. All query times were obtained using an implementation in the `C++` programming language compiled using the GNU C++ compiler with the optimizing option "-O2". All measurements were taken on an Intel Core i7-965 processor clocked at 3.2 GHz. We implemented Dijkstra's algorithm using binary heaps. Depending on the Dijkstra rank of a pair of vertices, the running time of a shortest path query ranges up to ≈ 35 ms.

7.3 Results regarding search accelerations

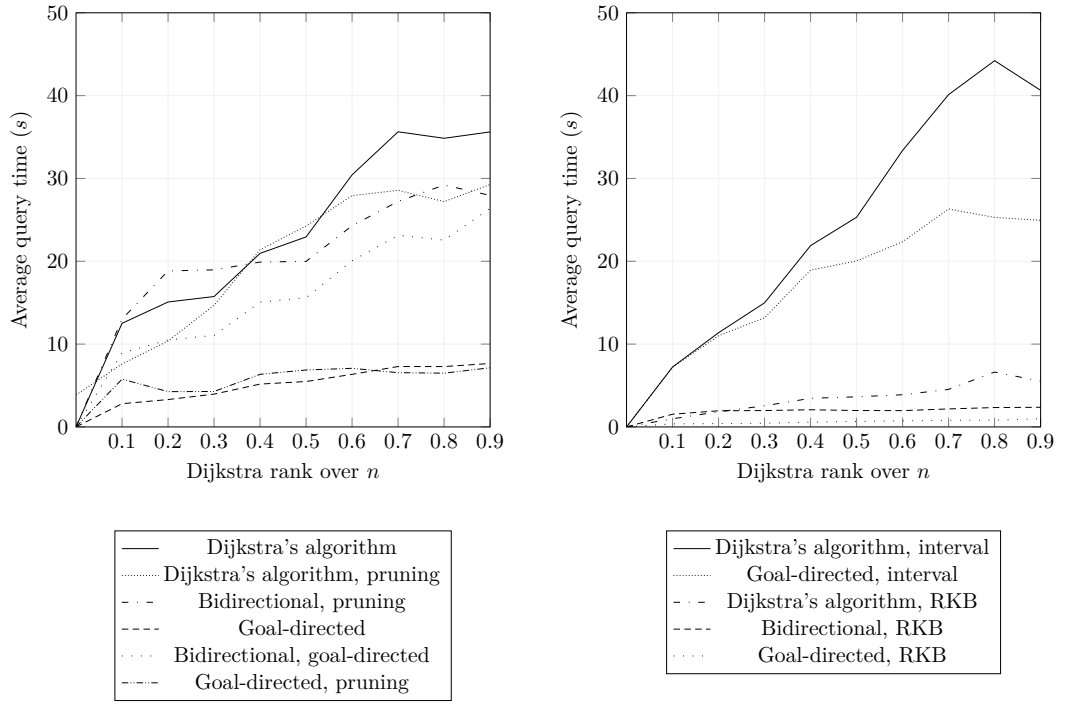
As a first step we evaluated the previously introduced approaches to accelerate individual searches without using the Divide and Conquer approach. The results are depicted in Figure 3a. We remark the following:

1. In order to achieve the best results regarding the goal-directed search we occasionally recompute the potential from scratch. Specifically, we keep track of how many vertices are settled during the recomputation of the potential as well as how many vertices are settled during each subsequent goal-directed search. If the latter amount is within a fraction of α of the former, we reuse the potential in the search to be carried out in the next iteration. Otherwise, we mark the potential to be recomputed during the next query. We found experimentally that a factor of $\alpha = 0.15$ yields the best results.
2. Regarding the bidirectional goal-directed search: We found that the best choice for the combined potential is the average of the two potentials computed during the forward and backward search respectively. Additionally, we found that in order to obtain more accurate potentials it is worth the effort to compute the entire search tree from s to t in the forward search and vice versa in the backward search.
3. Both improvements over Dijkstra's algorithm, the pruning and the goal-directed search, can be combined to speed up the computation even more.

Our findings show that while all approaches lead to reduced computation time, the goal-directed approaches works best, beating a plain evaluation using Dijkstra's algorithm by almost an order of magnitude.

7.4 Results regarding the Divide and Conquer approach

We proceed to consider the impact of the Divide and Conquer approach on the query time (results are shown in Figure 3b). Combining Dijkstra's algorithm with the generic Divide and Conquer approach (Algorithm 1) seems to have little effect on its own. Using the relaxed knapsack bound introduced in Subsection 6 however shows significant improvements. The combination of relaxed knapsack bounds and goal-direction yields the best results with a speedup factor ranging from 34 to 45 with an average of 38. A major contribution to this speed up is due to the fact that the Divide and Conquer approach cuts down on the required number of shortest path computations (see Figure 4): Dijkstra's algorithm alone requires $|\Theta|$-many shortest path computations regardless of the distance between source and target. The value is more than halved using the Divide and Conquer approach, it is cut down to less than ten percent if the relaxed knapsack bound is incorporated.



(a) Average query time for different search accelerations.

(b) Average query time for selected combinations of search accelerations together with the Divide and Conquer approach.

Figure 3 Average query time for different robust shortest path algorithms.

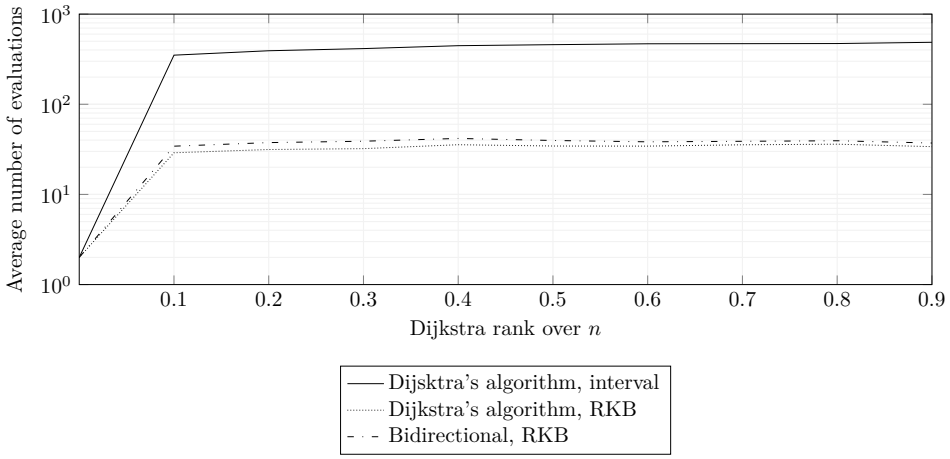


Figure 4 Average number of shortest path computations for different variants of the Divide and Conquer approach. The naive algorithm consistently requires $|\Theta| = 1,043$ evaluations.

8 Conclusion

We have presented an approximation scheme and a Divide and Conquer approach for general robust combinatorial optimization problems. The approximation scheme can be used to trade solution quality and running time. We introduced multiple techniques to accelerate the computation of robust shortest paths without abandoning solution quality ranging from the acceleration of individual queries to augmenting the Divide and Conquer approach by adding efficiently computable lower bounds of high quality. We evaluated our approaches by performing computational experiments on a digraph corresponding to a reasonable large road network. We found that a combination of the acceleration techniques decreased computation time by a factor of up to 45.

As the result for only Γ many shortest path computations does not hold and similar results seem unattainable in light of the counter-example, we give a currently best possible practical approach to solve the fundamental problem of shortest path in the classic Bertsimas Sim model for robustness.

References

1 Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 782–793. Society for Industrial and Applied Mathematics, 2010.

2 Rachit Agarwal, Matthew Caesar, Brighten Godfrey, and Ben Y. Zhao. Shortest paths in less than a millisecond. *CoRR*, abs/1206.1134, 2012. doi:10.1145/2342549.2342559.

3 Alper Atamtürk and Muhong Zhang. Two-stage robust network flow and design under demand uncertainty. *Oper. Res.*, 55(4):662–673, 2007. doi:10.1287/opre.1070.0428.

4 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. *Route Planning in Transportation Networks*, pages 19–80. Springer International Publishing, 2016. doi:10.1007/978-3-319-49487-6_2.

5 Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011. doi:10.1137/080734510.

6 Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1):49–71, 2003. doi:10.1007/s10107-003-0396-4.

7 Christina Büsing. The exact subgraph recoverable robust shortest path problem. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos Zaroliagis, editors, *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, pages 231–248. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-05465-5_9.

8 Christina Büsing. Recoverable robust shortest path problems. *Networks*, 59(1):181–189, 2012. doi:10.1002/net.20487.

9 Atish Das Sarma, Sreenivas Gollapudi, Marc Najork, and Rina Panigrahy. A sketch-based distance oracle for web-scale graphs. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 401–410. ACM, 2010. doi:10.1145/1718487.1718537.

10 Daniel Delling and Dorothea Wagner. Time-dependent route planning. *Robust and online large-scale optimization*, 5868(1):207–230, 2009. doi:10.1007/978-3-319-17885-1_101392.

11 Maged M Dessouky, Fernando Ordonez, and Ilgaz Sungur. A robust optimization approach for the capacitated vehicle routing problem with demand uncertainty. *IIE Transactions*, 40(5):509–523, 2008. doi:10.1080/07408170701745378.

12 Maciej Drwal. Complexity of scheduling on parallel identical machines to minimize total flow time with interval data and minmax regret criterion. *CoRR*, abs/1412.4273, 2014. URL: http://arxiv.org/abs/1412.4273.

13 Virginie Gabrel, Cécile Murat, and Aurélie Thiele. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 235(3):471–483, 2014. doi:10.1016/j.ejor.2013.09.036.

14 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In Catherine C. McGeoch, editor, *Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings*, pages 319–333. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-68552-4_24.

15 Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

16 Andrey Gubichev, Srikanta Bedathur, Stephan Seufert, and Gerhard Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 499–508. ACM, 2010. doi:10.1145/1871437.1871503.

17 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968. doi:10.1109/tssc.1968.300136.

18 Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast point-to-point shortest path computations with arc-flags. In *The Shortest Path Problem, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, November 13-14, 2006*, pages 41–72, 2006. doi:10.1090/dimacs/074/03.

19 Changhyun Kwon, Taehan Lee, and Paul Berglund. Robust shortest path problems with two uncertain multiplicative cost coefficients. *Naval Research Logistics (NRL)*, 60(5):375–394, 2013. doi:10.1002/nav.21540.

20 Supakom Mudchanatongsuk, Fernando Ordóñez, and Jie Liu. Robust solutions for network design under transportation cost and demand uncertainty. *Journal of the Operational Research Society*, 59(5):652–662, 2008. doi:10.1057/palgrave.jors.2602362.

21 Michael Poss. Robust combinatorial optimization with variable cost uncertainty. *European Journal of Operational Research*, 237:836–845, 2014. doi:10.1016/j.ejor.2014.02.060.

22 Muhong Zhang. Two-stage minimax regret robust uncapacitated lot-sizing problems with demand uncertainty. *Operations Research Letters*, 39(5):342–345, 2011. doi:10.1016/j.orl.2011.06.013.

A Proofs

We begin by giving the proof of Theorem 4, which was stated as follows:

▶ **Theorem 4.** *Let v be a vertex and $\theta < \theta'$ where $\theta, \theta' \in \Theta$. Let $P_\theta, P_{\theta'}$ be (s,v)-paths that are optimal with respect to c_θ respectively $c_{\theta'}$. Let*

$$\Gamma\theta + c_\theta(P_\theta) > \Gamma\theta' + c_{\theta'}(P_{\theta'}). \tag{11}$$

Then a robust shortest (s,t)-path is either attained for a value $\neq \theta$ or it does not contain v.

Proof. Assume for a contradiction a shortest robust (s,t)-path P is attained for θ and P contains v. P consists of two subpaths, i.e. P_θ and a path P_v leading from v to t. We let P' be the (s,t)-path which consists of $P_{\theta'}$ and P_v. We have:

$$\begin{aligned} C^\Gamma(\theta) &= \Gamma\theta + c_\theta(P_\theta) + c_\theta(P_v) \\ &> \Gamma\theta' + c_{\theta'}(P_{\theta'}) + c_\theta(P_v) \\ &\geq \Gamma\theta' + c_{\theta'}(P_{\theta'}) + c_{\theta'}(P_v) \\ &\geq \Gamma\theta' + c_{\theta'}(P') \end{aligned} \tag{16}$$

We have used here that $c_\theta \geq c_{\theta'}$ which follows from $\theta < \theta'$. This inequality implies that P' is a robust (s,t)-path which is shorter than P which is clearly a contradiction. ◀

We go on to present a more general variant of Theorem 5: We assume that we used a version of Dijkstra's algorithm with respect to reduced costs $c^\pi_{\theta_{\min}}$ obtained from a potential π computed while conducting a search for $c^{\text{opt}}(\theta_{\min})$. During the execution of the search we settled vertices and obtained information regarding the shortest paths for the part of the graph we have explored: In a most general situation, this information is accessible via a fixed arc set $M \subseteq A$ and various subsets $B \subseteq M$ together with bounds $\lambda(B)$ fulfilling

$$\lambda(B) \leq c^\pi_{\theta_{\min}}(P \cap M) \,\forall\, P \in \mathcal{P}(s,t) \text{ with } B \subseteq P. \tag{17}$$

We give some examples for this abstract setting, but first observe that M should contain the arc set corresponding to some $s-t$ cut to yield a bound $\lambda(\emptyset) > 0$. Otherwise the right hand side of the inequality (17) is equal to 0 for some path P with $P \cap M = \emptyset$. In case that a shortest path search completes, it determines $c^{\text{opt}}_\pi(t)$ as the length of a shortest (s,t)-path for $c^\pi_{\theta_{\min}}$, which leads to $c^{\text{opt}}(\theta_{\min}) = c^{\text{opt}}_\pi(t) + \pi(s) - \pi(t)$. This allows us to infer $\lambda(\emptyset) = c^{\text{opt}}(\theta_{\min}) - \pi(s) + \pi(t)$ for the set M containing all scanned arcs. As before we let $M_{\theta_{\min}} \subseteq M$ be the restriction to arcs a with $d(a) > \theta_{\min}$.

▶ **Example 7.** If we stop unidirectional search prematurely we can use for M the set of arcs, that have a head with settled label and $\lambda(\emptyset)$ can be chosen as the last settled distance label from the search. This situation applies to Theorem 5. Additionally, for some arc $a = (u,v) \in M_{\theta_{\min}}$ we can infer $\lambda(\{a\})$ as the label that v received from u via a because it is a lower bound on $c^\pi_{\theta_{\min}}(P \cap M)$ for any (s,t)-path P that contains a.

▶ **Example 8.** If some bidirectional Dijkstra search has been stopped prematurely, then let M^s be the set of arcs that have their head settled by the search from s, and let M^t contain the arcs with their tail settled by the search from t. We can use $M = M^s \cup M^t$ and for $\lambda(\emptyset)$ the sum of both lastly settled distance labels in the searches from s and t. For some $B = \{e_s, e_t\}$ $e_s \in M^s$, $e_t \in M^t$, $e_s \neq e_t$ we get for $\lambda(B)$ the sum of the head label from e_s, the tail label from e_t, and both arc costs $c^\pi_{\theta_{\min}}(e_s) + c^\pi_{\theta_{\min}}(e_t)$. Similar bounds for singleton B can be derived as well.

The idea of Theorem 9 is to compute a bound for $c^{\text{opt}}(\theta_0)$ using the abstract bound information. In a suitable program we optimize over the arcs in $M_{\theta_{\min}}$ that an imaginary path P could contain to minimize $c_{\theta_0}(P)$. The program also makes use of values $c^{\text{opt}}(\theta')$ known from previous computations if $\theta' > \theta$.

▶ **Theorem 9.** *Given a potential π, an arc set M, and a collection $\mathcal{B} \subset 2^{M_{\theta_{\min}}}$, such that bounds $\lambda(B)$ fulfilling* (17) *can be obtained for each B in $\mathcal{B}$, then for each $\theta_0 > \theta_{\min}$, such*

that π is also feasible for c_{θ_0}, we obtain a bound $O_{\theta_0} \leq c^{opt}(\theta_0)$ where O_{θ_0} is an optimum of

$$\begin{aligned}
&\textit{min} && y - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_0) - \theta_{\min}) && && (18a)\\
&\text{s.t.} && (\lambda(B) + \pi(s) - \pi(t)) \prod_{b \in B} x_b \leq y && \forall B \in \mathcal{B} && (18b)\\
& && y - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta') - \theta_{\min}) \geq c^{opt}(\theta') && && \\
& && && \forall \theta' : \theta_0 < \theta' \textit{ with } c^{opt}(\theta') \textit{ known} && (18c)\\
&\textit{variables:} && y \geq 0,\ x \in \{0,1\}^{M_{\theta_{\min}}} && && (18d)
\end{aligned}$$

Proof. Let P be any (s,t)-path. We show that $c_{\theta_0}(P) \geq O_{\theta_0}$ holds: Let us consider the arc sets $P', \bar{P} \subseteq P$ given by $P' \coloneqq P \cap M$ and $\bar{P} \coloneqq P \setminus P'$.

We claim that setting $x_a \coloneqq 1$ if and only if $a \in P' \cap M_{\theta_{\min}}$ together with $y \coloneqq c_{\theta_{\min}}(P') + c_{\theta_0}(\bar{P})$ constitutes a feasible solution to (18) and the cost of this solution is then a lower bound on $c_{\theta_0}(P)$. To get the lower bound we can first write $c_{\theta_0}(P')$ in terms of $c_{\theta_{\min}}(P')$:

$$\begin{aligned}
c_{\theta_0}(P') &= c(P') + \sum_{a \in P' : d(a) > \theta_{\min}} \max\{d(a) - \theta_0, 0\} + c_{\theta_{\min}}(P') - c_{\theta_{\min}}(P')\\
&= c(P') + \sum_{a \in P' : d(a) > \theta_{\min}} \max\{d(a) - \theta_0, 0\} + c_{\theta_{\min}}(P')\\
&\quad - \left(c(P') + \sum_{a \in P' : d(a) > \theta_{\min}} \max\{d(a) - \theta_{\min}, 0\} \right)\\
&= c_{\theta_{\min}}(P') + \sum_{a \in P' : d(a) > \theta_{\min}} (\max\{d(a) - \theta_0, 0\} - \max\{d(a) - \theta_{\min}, 0\})\\
&= c_{\theta_{\min}}(P') - \sum_{a \in P' : d(a) > \theta_{\min}} (\min(d(a), \theta_0) - \theta_{\min})\\
&= c_{\theta_{\min}}(P') - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_0) - \theta_{\min})
\end{aligned} \tag{19}$$

Here, the last equality holds, because by its definition P' is fully contained in M and all of its arcs with $d(a) > \theta_{\min}$ are contained in $M_{\theta_{\min}}$. With this expression we obtain

$$\begin{aligned}
c_{\theta_0}(P) &= c_{\theta_0}(\bar{P}) + c_{\theta_0}(P')\\
&= c_{\theta_0}(\bar{P}) + c_{\theta_{\min}}(P') - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_0) - \theta_{\min})\\
&= y - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_0) - \theta_{\min})\\
&\geq O_{\theta_0}
\end{aligned} \tag{20}$$

where the last inequality only holds if x, y is a feasible solution of (18). To prove this feasibility, we first consider (18b) and let $B \in \mathcal{B}$. If $B \nsubseteq P \cap M_{\theta_{\min}}$ then the corresponding Inequality (18b) has its left hand side equal to zero by the definition of x and is feasible. So let $B \subseteq P \cap M_{\theta_{\min}}$ which means that $\prod_{b \in B} x_b = 1$. Feasibility of (18b) in this case then

follows from the feasibility of π for c_{θ_0}, we first have:

$$\begin{aligned} y &= c_{\theta_{\min}}(P') + c_{\theta_0}(\bar{P}) \\ &= \sum_{a=(u,v)\in P'} (c^{\pi}_{\theta_{\min}}(a) + \pi(u) - \pi(v)) + \sum_{a=(u,v)\in \bar{P}} (c^{\pi}_{\theta_0}(a) + \pi(u) - \pi(v)) \\ &\geq \sum_{a=(u,v)\in P'} (c^{\pi}_{\theta_{\min}}(a) + \pi(u) - \pi(v)) + \sum_{a=(u,v)\in \bar{P}} (\pi(u) - \pi(v)) \\ &= c^{\pi}_{\theta_{\min}}(P') + \pi(s) - \pi(t) \end{aligned} \tag{21}$$

Here the last equality follows from resolving the telescope sum for the (s,t)-path $P = P' \cup \bar{P}$. Since $B \subseteq P \cap M$ we can use the bound $c^{\pi}_{\theta_{\min}}(P') = c^{\pi}_{\theta_{\min}}(P \cap M) \geq \lambda(B)$ which now implies (18b).

To show that Inequalities (18c) are satisfied, let $\theta' \geq \theta_0$ and $c^{\text{opt}}(\theta')$ be known. We know that $c^{\text{opt}}(\theta') \leq c_{\theta'}(P)$ because P is an (s,t)-path. So we are interested in bounding $c_{\theta'}(P) = c_{\theta'}(\bar{P}) + c_{\theta'}(P')$ against the left hand side of (18c). Because $\theta' > \theta_{\min}$ holds, we can do a similar calculation as before to express $c_{\theta'}(P')$ in terms of $c_{\theta_{\min}}(P')$:

$$c_{\theta'}(P') = c_{\theta_{\min}}(P') - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a), \theta') - \theta_{\min})$$

This implies

$$\begin{aligned} c^{\text{opt}}(\theta') &\leq c_{\theta'}(P) \\ &= c_{\theta'}(\bar{P}) + c_{\theta'}(P') \\ &= c_{\theta'}(\bar{P}) + c_{\theta_{\min}}(P') - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a), \theta') - \theta_{\min}) \\ &\leq c_{\theta_0}(\bar{P}) + c_{\theta_{\min}}(P') - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a), \theta') - \theta_{\min}) \\ &= y - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a), \theta') - \theta_{\min}) \end{aligned} \tag{22}$$

where the last inequality holds because $\theta' > \theta_0$ implies $c_{\theta'}(\bar{P}) \leq c_{\theta_0}(\bar{P})$. ◀

▶ **Theorem 10.** *For each $\theta_{\min} < \theta_0 < \theta_1$ such that π is also feasible for c_{θ_0} and c_{θ_1} we have $c^{opt}(\theta_{\min}) \geq O_{\theta_0} \geq O_{\theta_1} \geq c^{opt}(\theta')$ for all θ' that were considered in* (18c) *for both O_{θ_0} and O_{θ_1}.*

Proof. We consider the definitions of (18) for θ_0 and θ_1 respectively. Observe that the sets $M_{\theta_{\min}}$, the bounds $\lambda(B)$ as well as constraints (18b) and (18c) are independent of θ_0 and thus both programs for O_{θ_0} and O_{θ_1} optimize over the same set of feasible solutions. The only difference is the objective function, where for some $a \in M_{\theta_{\min}}$ its coefficient for θ_1 is less or equal than its coefficient for θ_0. This implies $O_{\theta_0} \geq O_{\theta_1}$ but also $c^{\text{opt}}(\theta_{\min}) \geq O_{\theta_0}$: Note that $O_{\theta_{\min}}$ is well-defined and contains only variable y because $M_{\theta_{\min}} = \emptyset$. An optimum is given by $y = c^{\text{opt}}(\theta_{\min})$ and thus $c^{\text{opt}}(\theta_{\min}) \geq O_{\theta_{\min}} \geq O_{\theta_0}$ because $\theta_0 > \theta_{\min}$. Finally, it holds for some θ' which was considered in (18b), that

$$\begin{aligned} O_{\theta_1} &= y - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a), \theta_1) - \theta_{\min}) \\ &\geq y - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a), \theta') - \theta_{\min}) \\ &\geq c^{\text{opt}}(\theta'). \end{aligned}$$

◀

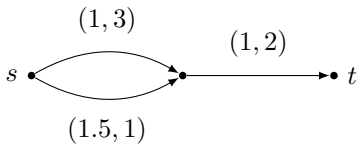


Figure 5 A counter-example to a claim regarding robust combinatorial optimization. Numbers on arcs represent costs and deviations.

B A counter-example to a claim regarding robust combinatorial optimization

We consider a claim made in [21] regarding a type of combinatorial optimization problems solvable by a dynamic programming (DP) algorithm. A combinatorial optimization problem is solvable by a DP algorithm if it can be expressed using a set of functional equations. More specifically, it is assumed that there is a set of states denoted by S with a subset $\mathcal{O}$ of initial states and a final state N. The optimal cost of state $s \in S$ is given by $F(s)$, the set of variables set to 1 in state s is denoted by $q(s)$. The state $p(s,i) \in S$ is set to be the previous state of s where s is obtained from $p(s,i)$ by fixing variable $i \in q(s)$ to 1. The relationship between the states is assumed to be governed by the following set of functional equations:

$$\begin{cases} F(s) = \min_{i \in q(s)} \{F(p(s,i)) + c_i\}, & s \in S \setminus \mathcal{O} \\ F(s) = 0, & s \in \mathcal{O} \end{cases} \tag{23}$$

In order to solve this problem the functional equation is applied to determine the optimal cost of new states until the optimal cost of the final state is determined. The question is whether the robust counterpart of such a problem can be solved in a similar manner using functional equations.

▶ **Theorem 11** (Theorem 6 in the original article). *Consider an instance of a combinatorial optimization problem which can be solved in $\mathcal{O}(\tau)$ for some $\tau : \mathbb{N} \to N$ by using the functional equations* (23)*. Then, its robust version can be solved in $\mathcal{O}(\Gamma\tau)$ using the following functional equations:*

$$\begin{cases} F(s,\alpha) & = \min_{i \in q(s)} \{\max(F(p(s,i),\alpha) + c_i, F(p(s,i),\alpha-1) + c_i + d_i)\}, \\ & \qquad\qquad s \in S \setminus \mathcal{O}, 1 \le \alpha \le \Gamma \\ F(s,0) & = \min_{i \in q(s)} \{F(p(s,i),0) + c_i\}, & s \in S \setminus \mathcal{O} \\ F(s,\alpha) & = 0, & 0 \le \alpha \le \Gamma, s \in \mathcal{O} \end{cases} \tag{24}$$

As an example of such a problem the authors consider the shortest path in a directed graph with conservative arc costs. It is well known that in this case the Bellman-Ford algorithm finds a shortest path by solving a dynamic program. As a counter-example to the claim stated above, we consider the graph in Figure 5 together with a parameter of $\Gamma = 1$. It should be apparent, that the robust shortest path in this case is the lower path with a total cost of 4.5. In order to compute the shortest path we start evaluating the functional equations for $\alpha = 0$. In this the coefficients coincide with those of the original problem. The graph corresponding to these functional equations is shown in Figure 6. Unfortunately, the path resulting from applying the functional equations is the upper path which has total costs of 5. The failure is due to the fact that the equations do not take into account that the first arc on the upper path has a high value of d.

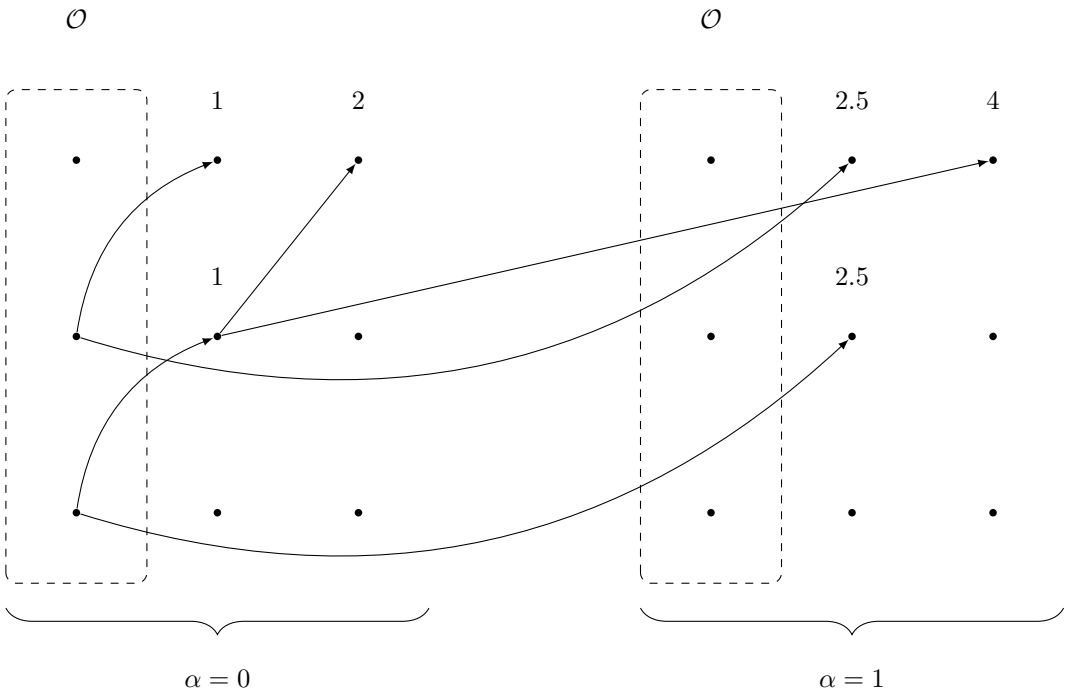


Figure 6 A depiction of the functional equations applied to the robust shortest path problem in Figure 5.

C Figures and tables

The following table contains the average query time plotted in Figures 3a and 3b. Regarding the distribution of the values: As is usually the case when it comes to the evaluation of running times, there is a certain variance in the recorded data. Figure 7 shows the distribution of running times for vertices with large Dijkstra ranks. Note that while the minimum / maximum query times are spread far apart, many of the individual values fall into much smaller intervals around the average. This behavior is consistent throughout the data and justifies the comparison based on the average query time.

Table 1 Average query time in seconds for various algorithms with respect to different ranks

	Dijkstra rank over n									
Algorithm	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Dijkstra's algorithm	0.00	12.51	15.08	15.74	20.95	22.94	30.43	35.63	34.85	35.62
Simple pruning	3.87	7.59	10.37	14.69	21.40	24.25	27.93	28.56	27.21	29.27
Bidirectional, pruning	0.00	13.02	18.84	18.96	19.90	19.99	24.29	27.23	29.20	27.95
Goal-directed	0.00	2.80	3.30	3.96	5.17	5.48	6.35	7.29	7.29	7.66
Bidirectional, goal-directed	0.01	8.93	10.50	11.06	15.06	15.61	20.06	23.14	22.55	26.38
Goal-directed, pruning	0.00	5.77	4.26	4.28	6.35	6.87	7.08	6.55	6.49	7.14
Dijkstra's algorithm, interval	0.00	7.23	11.37	14.98	21.89	25.31	33.38	40.12	44.22	40.64
Goal-directed, interval	0.00	7.21	11.01	13.19	18.92	20.04	22.36	26.31	25.29	24.94
Dijkstra's algorithm, RKB	0.01	0.98	1.83	2.56	3.45	3.64	3.88	4.53	6.62	5.53
Bidirectional, RKB	0.00	1.54	1.95	1.98	2.07	1.98	1.97	2.17	2.34	2.37
Goal-directed, RKB	0.02	0.37	0.40	0.44	0.58	0.67	0.74	0.80	0.82	0.99

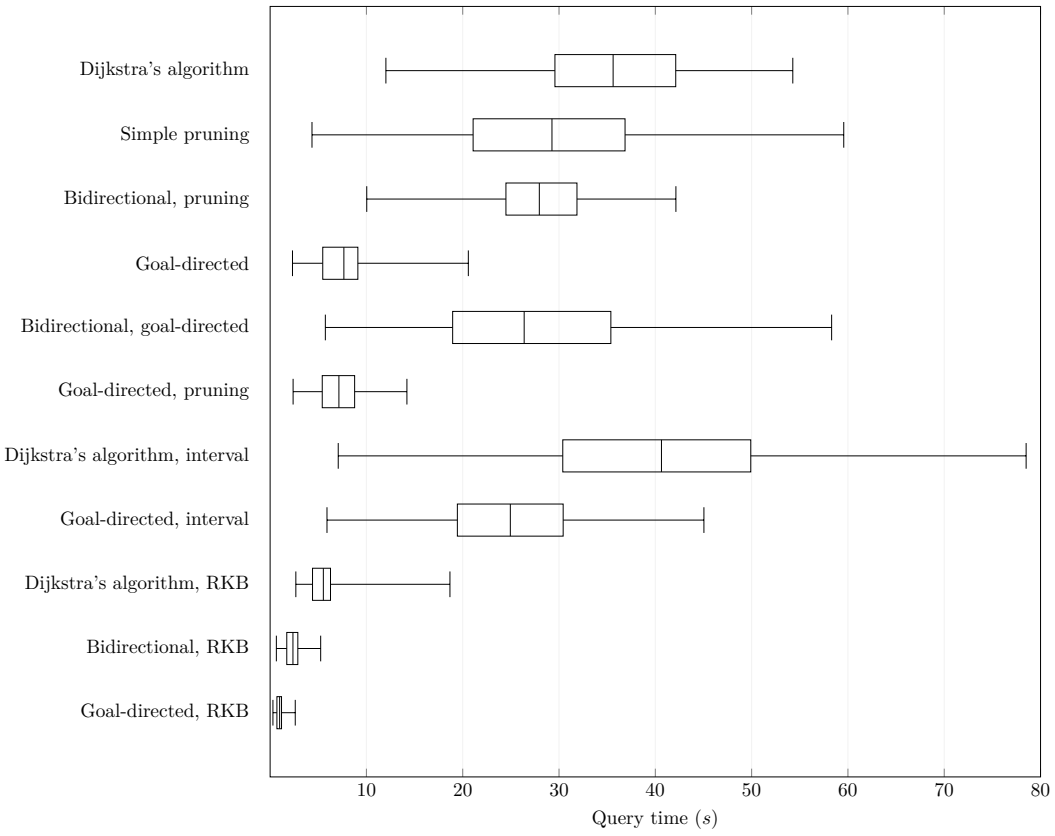


Figure 7 Distribution of the recorded running times. The boxes show minimum, first quartile, average, third quartile, and maximum for a rank of $0.9 \cdot n$.