European Journal of Operational Research 274 (2019) 671–686

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

Algorithms and uncertainty sets for data-driven robust shortest path problems

André Chassein[a], Trivikram Dokka[b], Marc Goerigk[c,*]

[a] Data Analytics Center of Excellence, Deutsche Post DHL Group, Bonn, Germany
[b] Department of Management Science, Lancaster University, Lancaster, United Kingdom
[c] Network and Data Science Management, University of Siegen, Germany

ARTICLE INFO

Article history:
Received 4 December 2017
Accepted 5 October 2018
Available online 11 October 2018

Keywords:
Robustness and sensitivity analysis
Robust shortest paths
Uncertainty sets
Data-driven robust optimization

ABSTRACT

We consider robust shortest path problems, where the aim is to find a path that optimizes the worst-case performance over an uncertainty set containing all relevant scenarios for arc costs. The usual approach for such problems is to assume this uncertainty set given by an expert who can advise on the shape and size of the set.

Following the idea of data-driven robust optimization, we instead construct a range of uncertainty sets from the current literature based on real-world traffic measurements provided by the City of Chicago. We then compare the performance of the resulting robust paths within and outside the sample, which allows us to draw conclusions on the suitability of uncertainty sets.

Based on our experiments, we then focus on ellipsoidal uncertainty sets, and develop a new solution algorithm that significantly outperforms a state-of-the-art solver.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

For classic shortest path problems in street networks, considerable speed-ups over a standard Dijkstra's algorithm have been achieved thanks to algorithm engineering techniques (Bast et al., 2016), which makes the use of real-time information even in large networks possible. Most types of robust shortest path problems, on the other hand, are NP-hard (see Yu & Yang, 1998), and real-time re-optimization has not been an option.

To formulate a robust problem, it is necessary to have a description of all possible and relevant scenarios that the solution should prepare against, the so-called uncertainty set. We refer to the surveys (Aissi, Bazgan, & Vanderpooten, 2009; Goerigk & Schöbel, 2016; Kasperski & Zieliński, 2016) for a general overview on the topic. The current literature on robust shortest paths usually assumes this set to be given, by some mixture of data-preprocessing and expert knowledge that is not part of the study. This means that different types of sets have been studied (compare, e.g., Büsing, 2012; Montemanni & Gambardella, 2004), but it has been impossible to address the question which would be the "right" choice.

A recent paradigm shift is data-driven robust optimization (see Bertsimas, Gupta, & Kallus, 2017), where building the uncertainty set from raw observations is part of the robust optimization problem. This paper is the first to follow such a perspective for shortest path problems. Based on real-world observations from the City of Chicago, we build a range of uncertainty sets, calculate the corresponding robust solutions, and perform an in-depth analysis of their performance. This allows us to give an indication which set is actually suitable for our application, and which are not.

In the second part of this paper, we then focus on the case of ellipsoidal uncertainty, and provide a branch-and-bound algorithm that is able to solve instances considerably faster than an off-the-shelf solver.

Parts of this paper were previously published as a conference paper in Dokka and Goerigk (2017). In comparison, we provide a completely new set of experimental results based on an observation period of 46 days (instead of one single day), which leads to a more detailed insight into the performance of different uncertainty sets. Furthermore, we provide a new analysis for axis-parallel ellipsoidal uncertainty sets, including an efficient branch-and-bound algorithm that is able to outperform CPLEX (CPLEX, 2014) by several orders of magnitude, pushing robust shortest paths towards applicability in real-time navigation systems.

* Corresponding author.
E-mail addresses: chassein@mathematik.uni-kl.de (A. Chassein), t.dokka@lancaster.ac.uk (T. Dokka), marc.goerigk@uni-siegen.de (M. Goerigk).

https://doi.org/10.1016/j.ejor.2018.10.006
0377-2217/© 2018 Elsevier B.V. All rights reserved.

The remainder of the paper is structured as follows. In Section 2, we briefly introduce the robust shortest path problem along with six uncertainty sets used in this study. The experimental setup and results on real-world data are then presented in Section 3. Section 4 presents our algorithm for ellipsoidal uncertainty sets, and includes additional computational results. The paper is concluded in Section 5.

2. Uncertainty sets for the shortest path problem

In this section, we briefly introduce the robust shortest path problem and different approaches to model uncertainty sets that are used in the current literature. In the classic shortest path problem, we are given a directed graph $G = (V, A)$ where V denotes the set of nodes, and A denotes the set of arcs. For every arc $e \in A$, we know its traversal time $c_e \geq 0$. For a start node s and a target node t, the aim is to find a path minimizing the total travel time given as the sum of times over all arcs that are part of the path.

More formally, we denote this problem as

$$\min \left\{ \boldsymbol{c}^t \boldsymbol{x} : \boldsymbol{x} \in \mathcal{X} \right\}$$

$\mathcal{X} \subseteq \{0, 1\}^n$ denotes the set of $s - t$-paths, and $n = |A|$ is the number of variables.

In our setting, we assume that travel times $\boldsymbol{c}$ are not known exactly. Instead, we are provided with a set $\mathcal{R}$ of travel time observ ations, where $\mathcal{R} = \{\boldsymbol{c}^1, \ldots, \boldsymbol{c}^N\}$ with $\boldsymbol{c}^i \in \mathbb{R}^n$. We also refer to $\mathcal{R}$ as the available raw data. Based on this data, an uncertainty set $\mathcal{U}$ is generated which is then used within the robust shortest path problem

$$\min \left\{ \max_{\boldsymbol{c} \in \mathcal{U}} \boldsymbol{c}^t \boldsymbol{x} : \boldsymbol{x} \in \mathcal{X} \right\}$$

that is, we search for a path that minimizes the worst-case costs over all costs in $\mathcal{U}$. We briefly sketch approaches to generate $\mathcal{U}$ in the following. Each is equipped with a scaling parameter to control its size (see also Chassein and Goerigk (2018) on the problem of choosing the size of an uncertainty set with a given shape). A visual example using four data points in two dimensions is provided for each approach in Fig. 1. We use the notation $[N] = \{1, \ldots, N\}$ and denote by $\hat{\boldsymbol{c}}$ the average of $\{\boldsymbol{c}^1, \ldots, \boldsymbol{c}^N\}$, i.e., $\hat{\boldsymbol{c}} = \frac{1}{N} \sum_{i \in [N]} \boldsymbol{c}^i$. For more details on the resulting models, we refer to the conference version of this paper (Dokka & Goerigk, 2017).

- Convex hull uncertainty (see, e.g., Kasperski & Zieliński, 2016; Yu & Yang, 1998): We set

  $$\mathcal{U}^{CH} = \mathcal{R}.$$

  Note that this is equivalent to using the convex hull of raw data. To scale this set, we substitute each point $\boldsymbol{c}^i$ with $\hat{\boldsymbol{c}} + \lambda(\boldsymbol{c}^i - \hat{\boldsymbol{c}})$ for a given $\lambda \geq 0$, and take the convex hull of the scaled data points.
- Interval uncertainty (see, e.g., Chassein & Goerigk, 2015): We set

  $$\mathcal{U}^I = \bigtimes_{i \in [n]} \left[\hat{c}_i + \lambda(\underline{c}_i - \hat{c}_i), \hat{c}_i + \lambda(\bar{c}_i - \hat{c}_i) \right]$$

  for some $\lambda \geq 0$, where $\bar{c}_i := \max_{j \in [N]} c_i^j$ and $\underline{c}_i := \min_{j \in [N]} c_i^j$. By $\bigtimes$, we denote the Cartesian product. Note that $\max_{\boldsymbol{c} \in \mathcal{U}^I} \boldsymbol{c}^t \boldsymbol{x} = \sum_{i \in [n]} (\hat{c}_i x_i + \lambda(\bar{c}_i - \hat{c}_i) x_i$ for every $\boldsymbol{x} \in \mathcal{X}$, and the robust optimization problem can be solved by solving a nominal (non-robust) problem. In Fig. 1(b), only the top right corner of the boxes becomes part of the robust problem.
- Ellipsoidal uncertainty (see, e.g., Ben-Tal & Nemirovski, 1998; Ben-Tal & Nemirovski, 1999): We set

  $$\mathcal{U}^E = \{\boldsymbol{c} : (\boldsymbol{c} - \hat{\boldsymbol{c}})^t \boldsymbol{\Sigma}^{-1} (\boldsymbol{c} - \hat{\boldsymbol{c}}) \leq \lambda\}$$

  with $\boldsymbol{\Sigma} = \frac{1}{N} \sum_{i \in [N]} (\boldsymbol{c}^i - \hat{\boldsymbol{c}})(\boldsymbol{c}^i - \hat{\boldsymbol{c}})^t$ derived from a maximum-likelihood fit of a normal distribution. The resulting sets are ellipsoids, see Fig. 1(c), containing the highest-density region of a fitted multivariate normal distribution.
- Budgeted uncertainty (see, e.g., Bertsimas & Sim, 2003; Bertsimas & Sim, 2004; Goerigk & Schöbel, 2016): We set

  $$\mathcal{U}^B = \left\{ \boldsymbol{c} : c_i = \hat{c}_i + (\bar{c}_i - \hat{c}_i)\delta_i \text{ for all } i \in [n], \quad \boldsymbol{0} \leq \boldsymbol{\delta} \leq \boldsymbol{1}, \quad \sum_{i \in [n]} \delta_i \leq \Gamma \right\}$$

  where the parameter Γ controls the size of $\mathcal{U}^B$. This approach can be seen as a variant of interval uncertainty, where the top right corner is cut off to decrease conservatism. The size of Γ controls the size of the region that is cut off from the box.
- Permutohull uncertainty (see Bertsimas & Brown, 2009): We set

  $$\mathcal{U}^{PH} = conv\left(\left\{ \sum_{i \in [N]} q_{\sigma(i)} \boldsymbol{c}^i : \sigma \in S_N \right\} \right)$$

  where S_N denotes the set of permutations on $[N]$, and $\boldsymbol{q}$ is a column of the matrix

  $$Q_N := \begin{pmatrix} 1 & \ldots & \frac{1}{N-2} & \frac{1}{N-1} & \frac{1}{N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \frac{1}{N-2} & \frac{1}{N-1} & \frac{1}{N} \\ 0 & \ldots & 0 & \frac{1}{N-1} & \frac{1}{N} \\ 0 & \ldots & 0 & 0 & \frac{1}{N} \end{pmatrix} \in \mathbb{R}^{N \times N}$$

  Scaling is included via the choice of the column, where using the jth column of Q_N corresponds to using the conditional value at risk $CVaR$ criterion with respect to risk level j/N. Using $\boldsymbol{q} = (\frac{1}{2}, \frac{1}{2}, 0, 0)$ as shown in Fig. 1(e) means that we take the convex hull of the average of each pair of original points; using $\boldsymbol{q} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0)$ amounts to taking the convex hull of the average of each triplet of original points.
- Symmetric permutohull uncertainty (see Bertsimas & Brown, 2009): As in the above case, but we generate $\mathcal{U}^{SPH}$ using columns of the matrix $\tilde{Q} \in \mathbb{R}^{N \times (\lfloor N/2 \rfloor + 1)}$ defined by

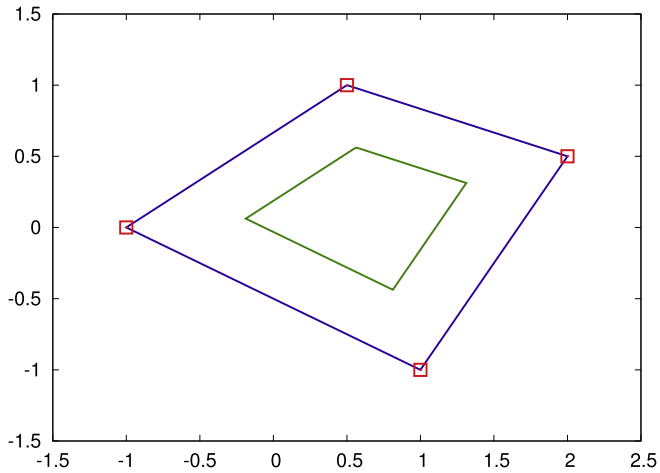
  $$\tilde{Q} := \frac{1}{N} \begin{pmatrix} 1 & 2 & 2 & \ldots & 2 \\ 1 & 1 & 2 & \ldots & 2 \\ 1 & 1 & 1 & \ldots & 2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & \ldots & 0 \\ 1 & 1 & 0 & \ldots & 0 \\ 1 & 0 & 0 & \ldots & 0 \end{pmatrix},$$

  instead, i.e., in the first column, all entries are $1/N$; in the second column, the first entry is $2/N$ and the last entry is 0, etc. The resulting sets are symmetric with respect to $\hat{\boldsymbol{c}}$.

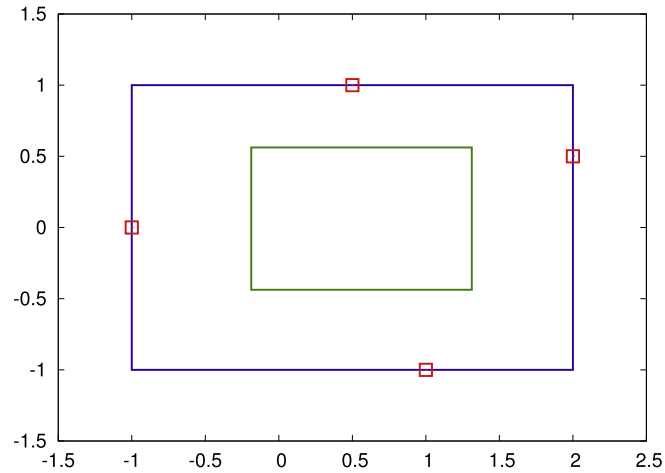
Table 1
Uncertainty sets in this study.

	$\mathcal{U}^{CH}$	$\mathcal{U}^I$	$\mathcal{U}^E$	$\mathcal{U}^B$	$\mathcal{U}^{PH}$	$\mathcal{U}^{SPH}$
Complexity	NPH	P	NPH	P	NPH	NPH
Model	IP	LP	MISOCP	MIP	MIP	MIP
Additional constraints	N	0	1	$n+1$	N^2	N^2
Additional variables	1	0	1	n	$2n$	$2n$

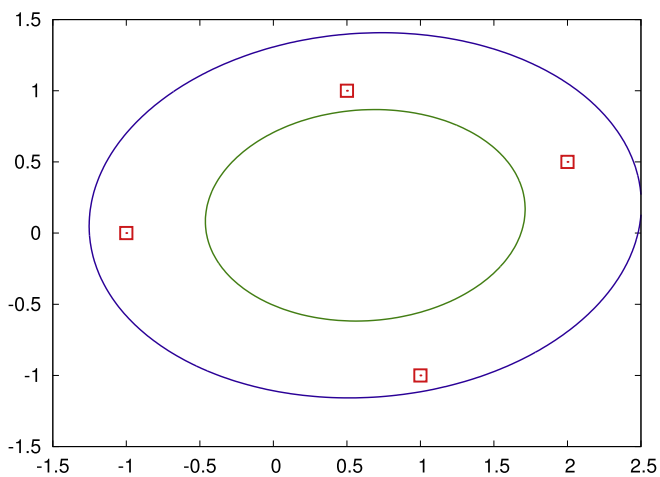
In total, we use six methods to generate uncertainty set $\mathcal{U}$ based on the raw data $\mathcal{R}$. The resulting optimization model and its complexity are summarized in Table 1. Here, "(M)IP" stands for



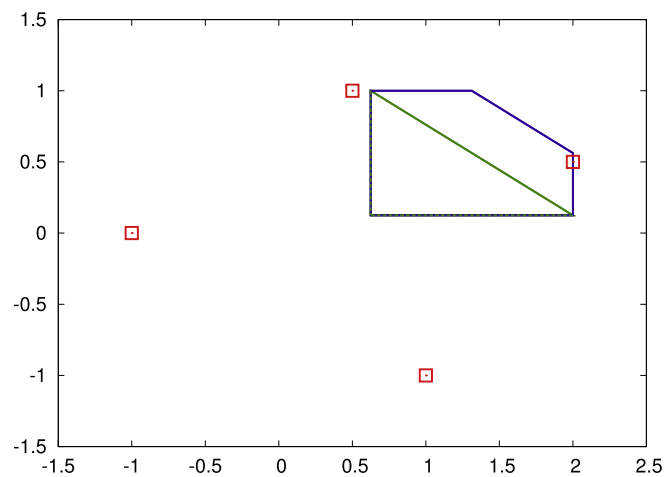
(a) Convex hull with $\lambda = 1$ and $\lambda = 0.5$.



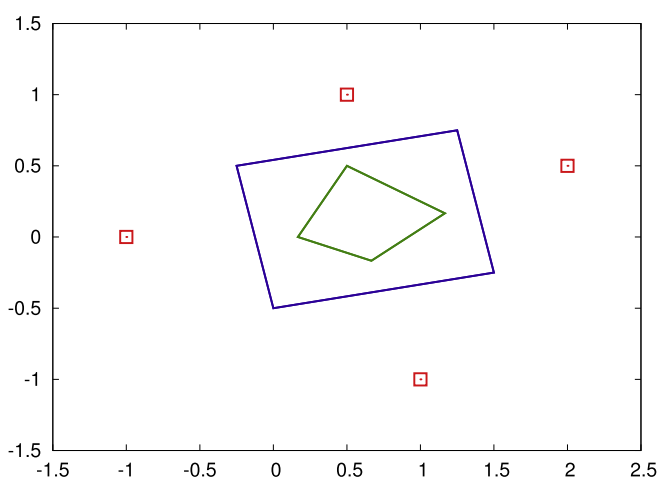
(b) Intervals with $\lambda = 1$ and $\lambda = 0.5$.



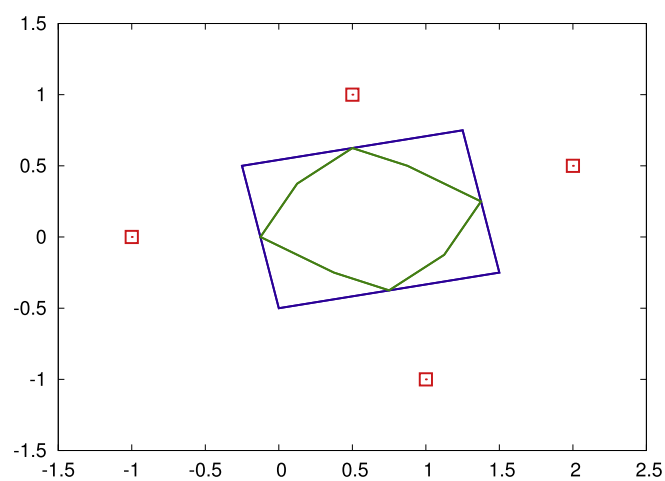
(c) Ellipsoid with $\lambda = 3$ and $\lambda = 1$.



(d) Budgeted uncertainty with $\Gamma = 1.5$ and $\Gamma = 1$.



(e) Permutohull uncertainty for $CVaR_{2/N}$ and $CVaR_{3/N}$ (i.e., $\boldsymbol{q} = (\frac{1}{2}, \frac{1}{2}, 0, 0)$ and $\boldsymbol{q} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0)$.



(f) Symmetric permutohull uncertainty for $\boldsymbol{q} = (\frac{1}{2}, \frac{1}{2}, 0, 0)$ and $\boldsymbol{q} = (\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, 0)$.

Fig. 1. Example uncertainty sets.

(mixed-)integer linear program, "LP" for linear program, and "MIS-OCP" for mixed-integer second order cone program. Furthermore, "NPH" denotes that the problem is NP-hard, while "P" means that it is solvable in polynomial time.

While the robust model with budgeted uncertainty sets can be solved in polynomial time using combinatorial algorithms, we still used the MIP formulation for our experiments, as it was sufficiently fast.

3. Real-world experiments

3.1. Data collection and cleaning

We used data provided by the City of Chicago[1], which provides a live traffic data interface. The data set consists of traffic updates for every 15-minute interval over a time horizon of 46 days spanning Tuesday morning, March 28, 2017 to Friday evening, May 12, 2017.

Out of all $46 \cdot 96 = 4416$ potential observations, only 4363 had usable data or were recorded due to server downtimes. Every data point contains the traffic speed for a subset of a total of 1257 segments (that is, a part of a street given through its geographical coordinates). The resulting plot in Fig. 2(a) with a zoom-in for the city center. The complete travel speed data set contains a total of 3,891,396 records. There were 1045 segments where the data was recorded at least once in the 4363 data points. For nearly 55% of the segments, at least 1340 data points were recorded, and more than 90% of them have at least 450 data points. For almost all segments at least 400 data points were recorded. We used linear interpolation to fill the missing records keeping in mind that data was collected over time. For segments that did not have any data, we set the travel speed to 20 miles per hour (which is slightly slower than the average speed in the network). Any speed record below 3 miles per hour was set to 3 miles per hour to ensure reasonable travel times. Segment lengths were given through longitude and latitude coordinates, and approximated using the Euclidean distance.

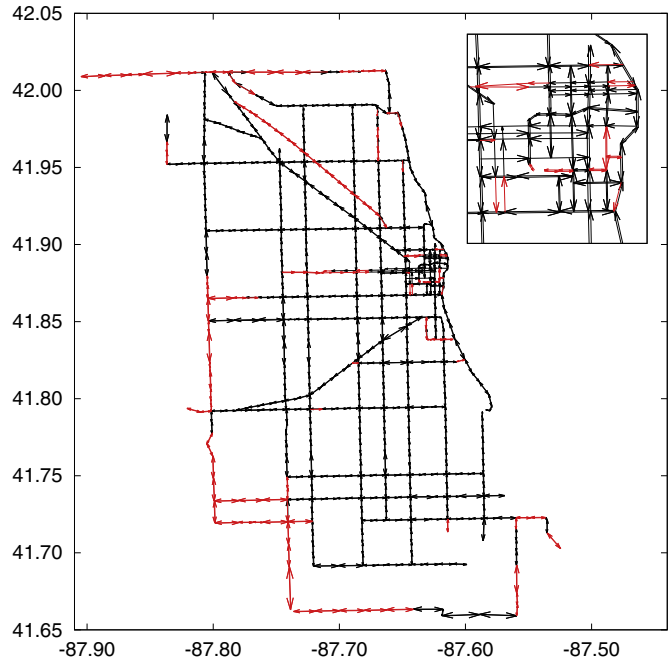
As segments are purely geographical objects without structure, we needed to create a graph for our experiments. To this end, segments were split when they crossed or nearly crossed, and start- and end-points that were sufficiently close to each other were identified as the same node. The resulting graph is shown in Fig. 2(b); note that this process slightly simplified the network, but kept its structure intact. The final graph contains 538 nodes and 1308 arcs. Using arc length and speed, we calculate their respective traversal time for each of the 4363 data points. In the following, we refer to 4363 *scenarios* generated this way.

We then used this full dataset to derive the following subsets aimed at providing robust solutions in different contexts:

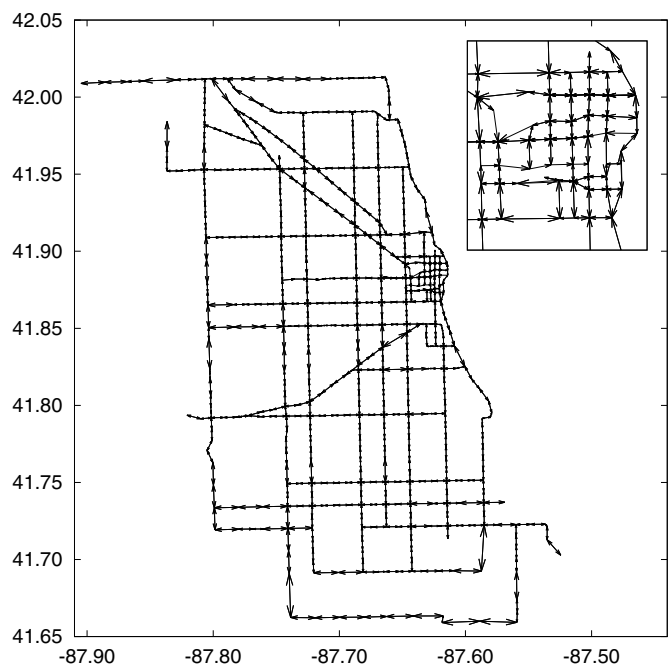
1. Find a path that is robust when driving during morning rush hours. We only use scenarios sampled on weekdays from 8am to 10am. These are 271 such scenarios ("mornings dataset").
2. Find a path that is robust when no additional information is given. We use all 4363 scenarios ("complete dataset").

In the following, we present results only for the mornings dataset. Results for the complete dataset can be found in Appendix A. Additionally, we also carried out experiments on an evenings dataset (272 scenarios for weekdays from 4pm to 6pm), a Tuesdays dataset (671 scenarios for all Tuesdays), and a weekends dataset (1141 scenarios for all Saturdays and Sundays). As results were found to be very similar, they are not presented here. All dataset are available from the corresponding author upon request.

[1] https://data.cityofchicago.org



(a) Raw segments with zoom-in for the city center, longitude versus latitude. In red are segments without data.



(b) Resulting graph model with zoom-in for the city center, longitude versus latitude.

Fig. 2. Chicago instance.

3.2. Data exploration

Fig. 3 visualizes the travel time data used in these experiments plotted against the time of one week, where each point represents the average travel speed over all segments in the network in one observation. The red line shows the hourly average travel speed, and the shaded area represents the corresponding 95% confidence band.

We find that there is little correlation between edges (for 99.36% of pairs of edges, the absolute value of correlation is ≤ 0.3),

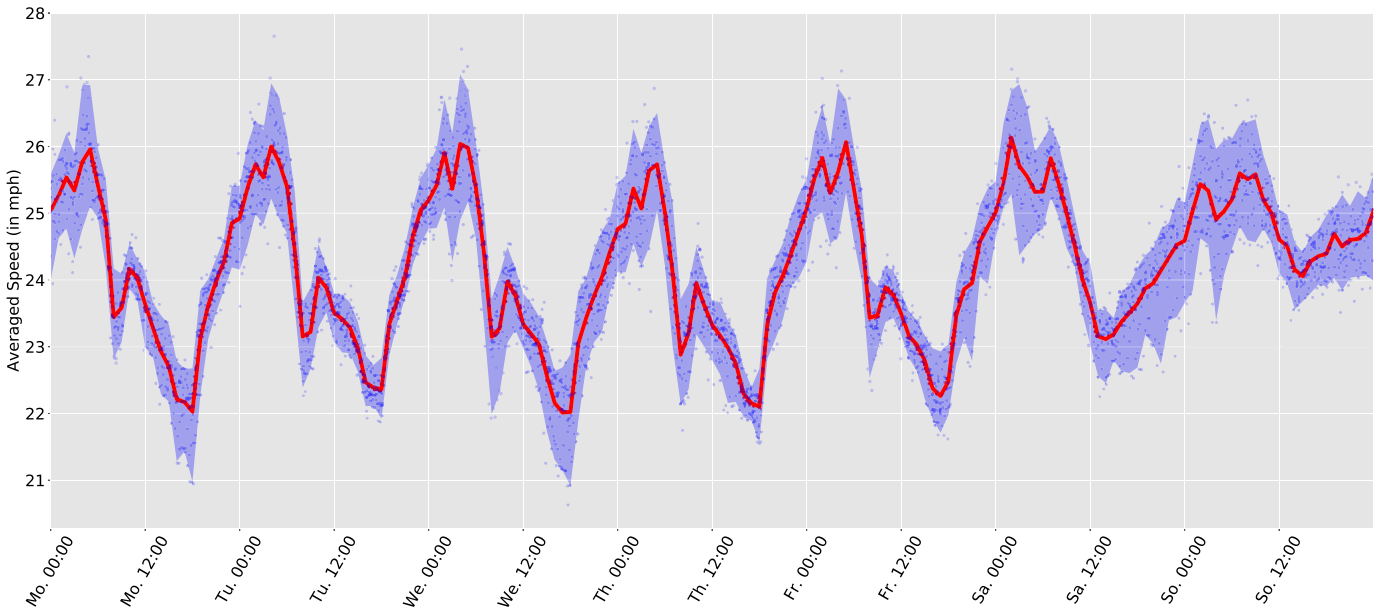


Fig. 3. Averaged recorded travel speed data. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

but a strong correlation between points in time (for 87.52% pairs of times, the absolute value of correlation is >0.6). This gives rise to the question whether optimal paths between fixed pairs of nodes change over time (if not, a robust optimization approach was not necessary). To consider this aspect, we generated 200 random $s-t$ pairs and calculated an optimal path for each scenario and pair. For only 6 out of 200 pairs, there was one path which was optimal in every scenario. For 128 pairs, there were at least 10 unique shortest paths; and for 63 pairs, there were at least 50 such paths. On average, there are 64.15 different shortest paths per $s-t$ pair.

3.3. Setup

Each uncertainty set can be tuned with a scaling parameter. For each parameter we generated 20 possible values, reflecting a reasonable range of choices for a decision maker:

- For $\mathcal{U}^{CH}$ and $\mathcal{U}^{I}$, $\lambda \in \{0.05, 0.10, \ldots, 1.00\}$.
- For $\mathcal{U}^{E}$, $\lambda \in \{0.5, 1.0, \ldots, 10.0\}$.
- For $\mathcal{U}^{B}$, $\Gamma \in \{1, 2, \ldots, 20\}$.
- For $\mathcal{U}^{PH}$, we used columns $\boldsymbol{q}_1, \boldsymbol{q}_3, \ldots, \boldsymbol{q}_{39}$.
- For $\mathcal{U}^{SPH}$, we used columns $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_{20}$.

Additionally, we calculate a solution to the average-case scenario $\hat{\boldsymbol{c}}$. Note that this is a special case of all uncertainty sets presented here if the scaling parameter is sufficiently small. Each uncertainty set is generated using 75% of scenarios sampled uniformly (e.g., 203 out of 271), and we evaluate solutions in-sample and out-sample separately. Furthermore, we generated 200 random $s-t$ pairs uniformly over the node set, and used each of the $6 \cdot 20$ methods on the same 200 pairs. Each of our 120 methods, hence, generates $200 \cdot 271 = 54,200$ objective values for the mornings set.

It is non-trivial to assess the quality of these robust solutions, see Chassein and Goerigk (2016b). If one just uses the average objective value, as an example, then one could as well calculate the solution optimizing the average scenario case to find the best performance with respect to this measure. To find a balanced evaluation of all methods, we used three performance criteria. Let $\mathcal{X}' = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^P\}$ with $P = 200$ be the set of paths generated by some solution approach.

- The average objective value over all $s-t$ pairs and all scenarios, i.e., the value

$$AV = \frac{1}{P} \sum_{i \in [P]} \left(\frac{1}{N} \sum_{j \in [N]} (\boldsymbol{c}^j)^t \boldsymbol{x}^i \right)$$

- The average of the worst-case objective value for each $s-t$ pair, i.e., the value

$$MAX = \frac{1}{P} \sum_{i \in [P]} \left(\max_{j \in [N]} (\boldsymbol{c}^j)^t \boldsymbol{x}^i \right)$$

- The average value of the worst 5% of objective values for each $s-t$ pair (as in the *CVaR* measure). Formally, let $C^i_j = \sum_{j \in [N]} (\boldsymbol{c}^j)^t \boldsymbol{x}^i$, and let $C^i_{(j)}$ be a sorting of indices such that $C^i_{(j)} \geq C^i_{(j+1)}$ for all $j \in [N-1]$. Then we use

$$CVaR = \frac{1}{P} \sum_{i \in [P]} \left(\frac{20}{N} \sum_{j \in [N/20]} C^i_{(j)} \right)$$

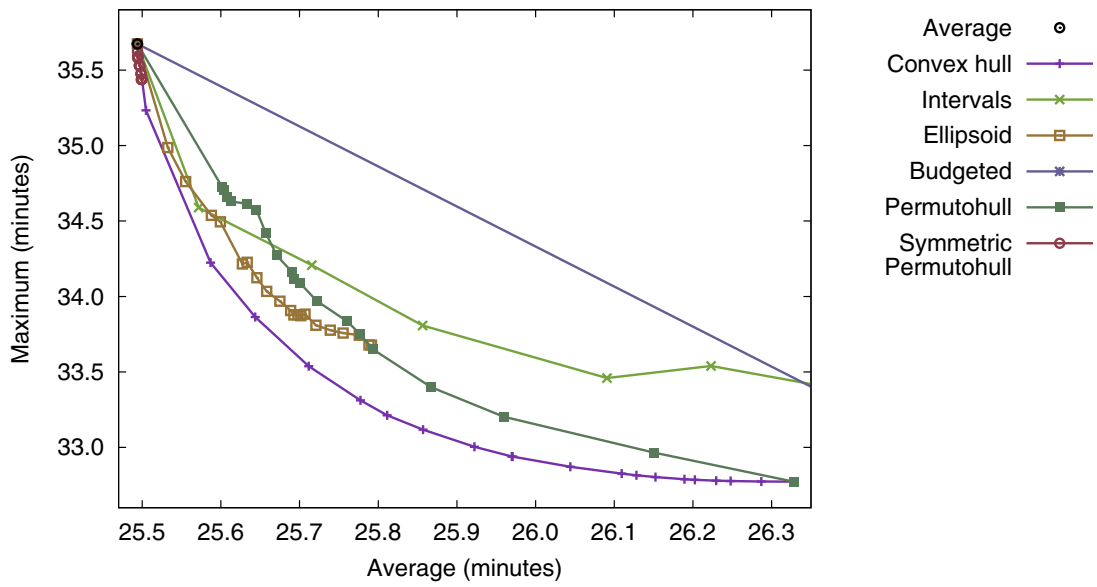
Note that many more criteria would be possible to use.

For all experiments we used a computer with a 16-core Intel Xeon E5-2670 processor, running at 2.60 gigahertz with 20 megabytes cache, and Ubuntu 12.04. Processes were pinned to one core. We used CPLEX v.12.6 to solve all problem formulations (note that specialized combinatorial algorithms are available for some problems).

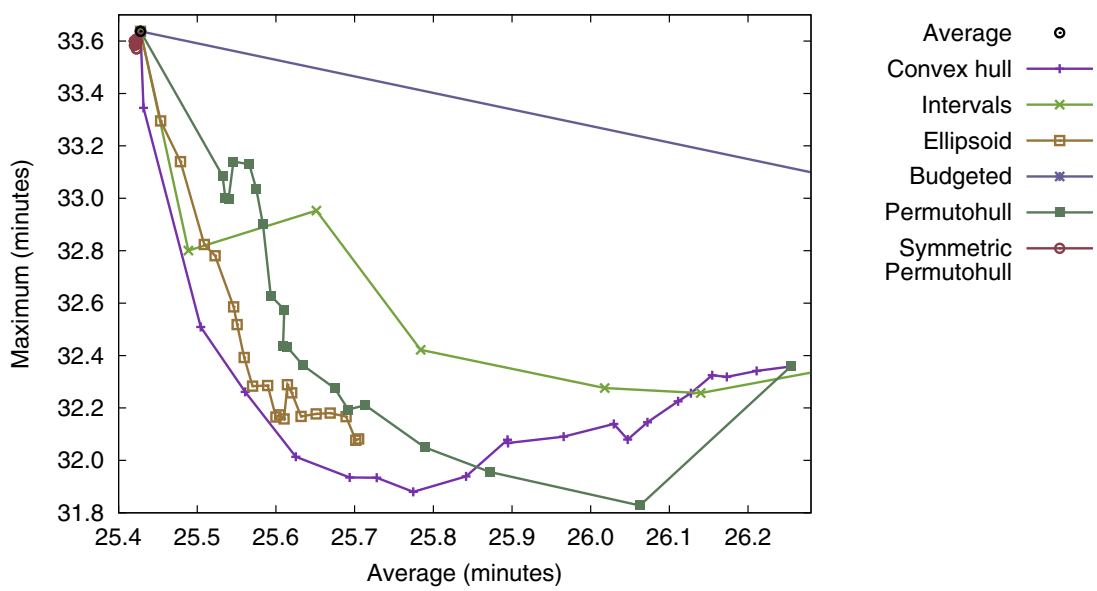
3.4. Results

We present the performance of solutions in Figs. 4 and 5. In each plot, the 20 parameter settings that belong to the same uncertainty set are connected by a line, including the average case as a 21st point. They are complemented with Fig. 6 showing the total computation times for the methods over all 200 shortest path calculations. For better visibility, Figs. 4 and 5 do not show outlying data; plots showing all data can be found in Appendix A.

The first set of plots in Fig. 4 shows the trade-off between the average and the maximum objective value; the second set of plots in Fig. 5 shows the trade-off between the average and the average of the 5% worst objective values. For each case, the in-sample and



(a) in-sample



(b) out-sample

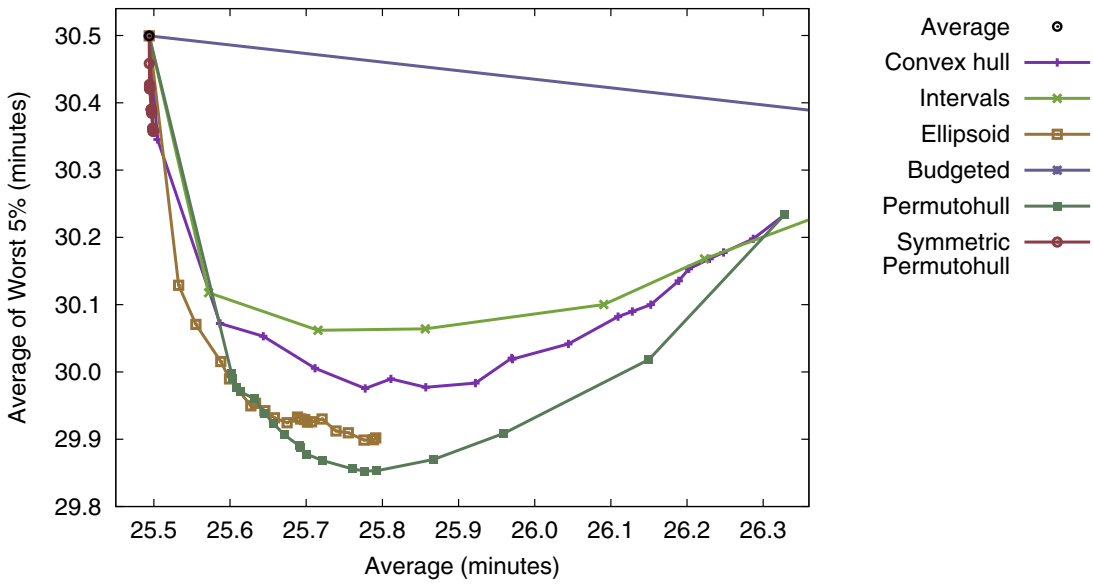
Fig. 4. Average vs. worst-case performance.

out-sample performance is shown (i.e., the performance on the set of scenarios that were used in building the uncertainty set, and the performance on the set of scenarios which were not used). All values are in minutes of travel time. Note that for all performance measures, smaller values indicate a better performance – hence, good trade-off solutions should move from the top left to the bottom right of the plots. In general, the points corresponding to the parameter settings that give weight to the average performance are on the left sides of the curves, while the more robust parameter settings are on the right sides, as would be expected.

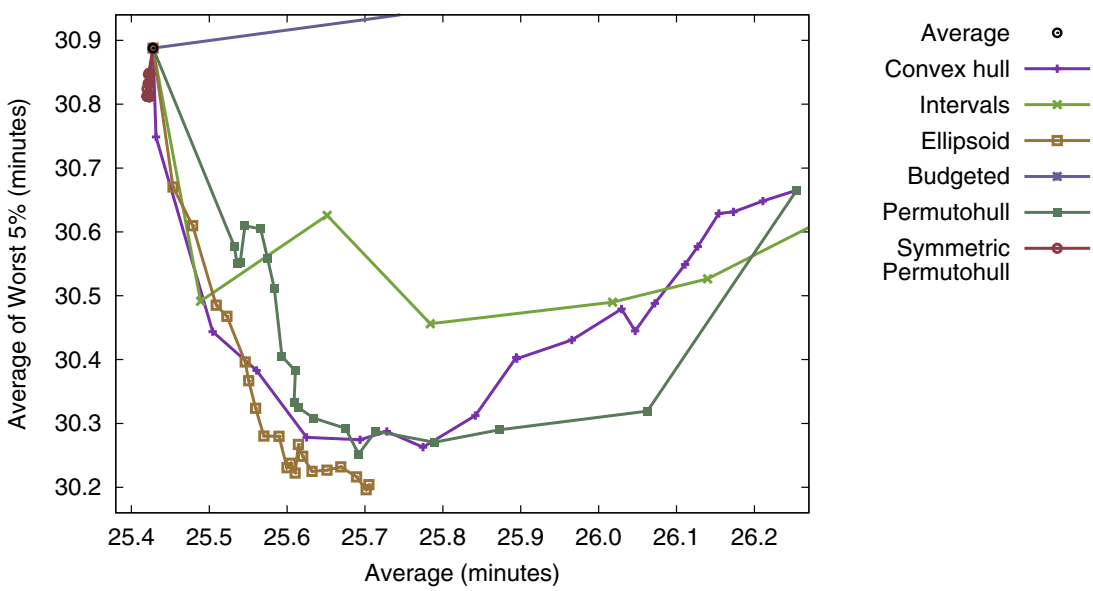
We first discuss the in-sample performance in Fig. 4(a). In general, we find that most methods indeed present a trade-off between average performance and robustness through their scaling parameter (i.e., curves run from top left to bottom right). Solutions calculated using the convex hull dominate the others. Symmetric permutohull solutions tend to focus on a good average performance, while ellipsoid and permutohull solutions show a broader front over the two criteria. Interval and budgeted uncertainty solutions tend to perform worse for larger scaling values, without the desired trade-off property, which confirms previous findings in Chassein and Goerigk (2016a).

Comparing these findings with the out-sample results in Fig. 4(b), we see that a general ranking of concepts is kept intact. The solutions generated by the convex hull lose their trade-off property, as they are apparently over-fitted to the data used in the sample (see also the results in Appendix A). We also find that the interval uncertainty outperforms budgeted uncertainty here, while they showed similar performance in-sample.

We now consider the results presented in Fig. 5. Here the average is plotted against the average performance of the 5% worst performing scenarios, averaged over all $s-t$ pairs. While the convex hull solutions showed the best trade-off in Fig. 4(a), we



(a) in-sample



(b) out-sample

Fig. 5. Average vs. CVaR performance.

find that the permutohull solutions are prominent among the non-dominated points in this case. As before, symmetric permutohull solutions tend to remain on one end of the spectrum with good average performance. Solutions obtained from ellipsoidal uncertainty are still among the best, and stable when considered out-sample (compare Fig. 5(a) and 5(b)). Interval uncertainty and in particular budgeted uncertainty do not perform well in comparison with the other approaches.

Regarding computation times (see Fig. 6), note that the two polynomially solvable approaches (intervals and budgeted) are also the fastest when using CPLEX; these computation times can be further improved using specialized algorithms. Using ellipsoids is faster than using the convex hull, which is in turn faster than using the symmetric permutohull. For the standard permutohull, the computation times are sensitive to the uncertainty set size; if the $\boldsymbol{q}$ vector that is used in the model has only few entries, computation times are smaller. This is in line with the intuition that the problem becomes easier if fewer scenarios need to be considered.

To summarize our findings in our experiment on the robust shortest path problem with real-world data:

- Convex hull solutions show good in-sample performance, but are not stable when facing scenarios out of sample.
- Interval solutions do not perform well in general, but are easy and fast to compute, which makes them a reasonable approach, in particular for smaller scalings.
- Budgeted uncertainty does not seem an adequate choice for robust shortest path problems. Scaling interval uncertainty sets gives better results and is easier to use and to solve.
- Ellipsoidal uncertainty solutions have good and stable overall performance and represent a large part of the non-dominated points in our results.

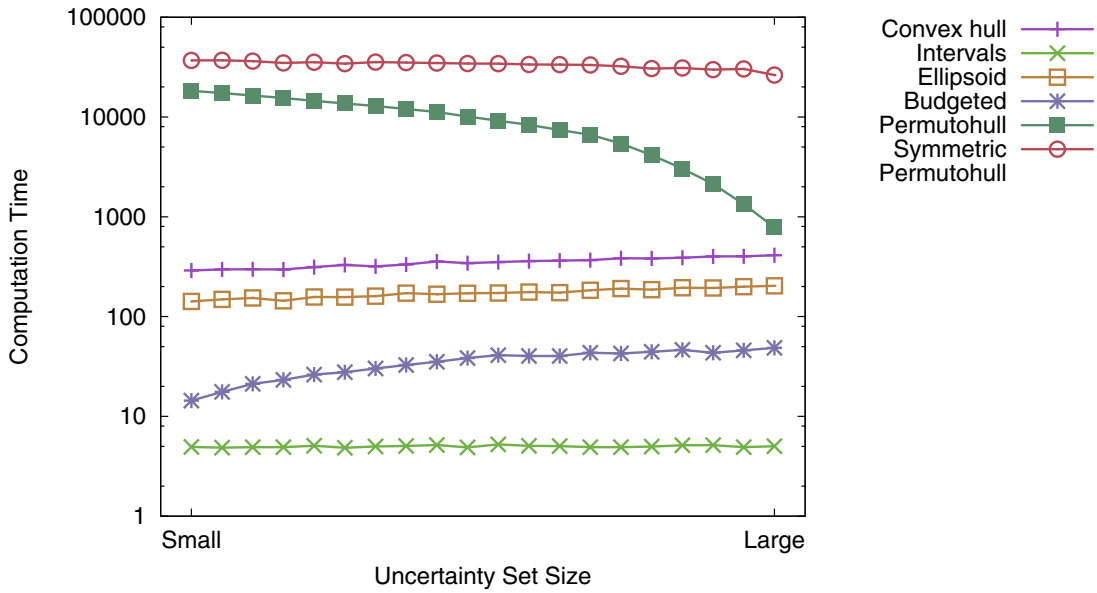


Fig. 6. Computation times in seconds.

- Permutohull solutions offer good trade-off solutions, whereas symmetric permutohull solutions tend to be less robust, but provide an excellent average performance. These methods also require most computational effort to find.

In the light of these findings, permutohull and ellipsoidal uncertainty tend to produce solutions with the best trade-off, while being computationally more challenging than most of the other approaches. The algorithmic research for robust shortest path problems with such structure should therefore be studied further. Results on additional experiments leading to the same conclusions can be found in the appendix. In the following section, we consider a variant of ellipsoidal uncertainty where correlation between arcs is ignored.

4. Ellipsoidal uncertainty sets

4.1. From general to axis-parallel ellipsoids

Since our experiments have shown that ellipsoidal uncertainty sets are a reasonable choice, we devote this section to these sets. First, we show that changing the general ellipsoid to an axis-parallel ellipsoid by setting all non-diagonal entries of $\boldsymbol{\Sigma}$ to 0, has almost no effect on the solutions found. Second, we derive a specialized branch-and-bound algorithm for such axis-parallel ellipsoidal uncertainty sets which clearly outperforms the standard approach of using a generic solver. Problems with the same structure were previously considered in Nikolova (2009), where a heuristic method was proposed.

Comparing general and axis-parallel ellipsoids for the experiments presented in Section 3, we find that the maximum deviation over all plotted datapoints is less than 0.002% for average travel times, less than 0.005% for average worst-case values, and less than 0.003% for average CVaR values. That is, plotted in our figures, general and axis-parallel ellipsoids would look indistinguishable. On the other hand, using axis-parallel ellipsoidal uncertainty sets in the robust model instead of general ellipsoids decreases the computation time significantly, see Fig. 7. The computation time can be further reduced by the use of specialized algorithms, as shown in the next section.

4.2. Efficient algorithm for axis-parallel ellipsoids

4.2.1. A bicriteria perspective

In this section, we describe an efficient branch-and-bound algorithm to solve the robust shortest path problem if the uncertainty set is given as an axis parallel ellipsoid. Recall that the mathematical formulation of the problem is

$$\begin{aligned} \min \quad & \hat{\boldsymbol{c}}^t \boldsymbol{x} + z \\ \text{s.t.} \quad & z^2 \geq \left(\boldsymbol{x}^t \boldsymbol{\Sigma} \boldsymbol{x}\right) \\ & \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

where $\boldsymbol{\Sigma}$ is a diagonal matrix specifying the shape and the size of the ellipsoid. Let $\boldsymbol{d}$ be the diagonal of $\boldsymbol{\Sigma}$. Since $\boldsymbol{x}$ is a binary vector, we have $x_i^2 = x_i$ and we can simplify the quadratic expression $\boldsymbol{x}^t \boldsymbol{\Sigma} \boldsymbol{x}$ to a linear expression $\boldsymbol{d}^t \boldsymbol{x}$. Hence the problem can be reduced to

$$\begin{aligned} \min \quad & \hat{\boldsymbol{c}}^t \boldsymbol{x} + \sqrt{\boldsymbol{d}^t \boldsymbol{x}} \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

As pointed out in Nikolova (2009), this problem can be transformed to the following bicriteria optimization problem.

$$\begin{aligned} \min \quad & \begin{pmatrix} \hat{\boldsymbol{c}}^t \boldsymbol{x} \\ \boldsymbol{d}^t \boldsymbol{x} \end{pmatrix} \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

There it is shown that each optimal solution of the robust problem is an efficient extreme solution of this bicriteria optimization problem. We call a solution $\boldsymbol{x}^*$ of this bicriteria optimization problem *efficient extreme* if there exists α_0 and α_1 with $0 \leq \alpha_0 < \alpha_1 \leq 1$ such that for all $\alpha \in [\alpha_0, \alpha_1]$ it holds that there exists no other solution $\boldsymbol{x}'$ with $(\alpha \boldsymbol{c} + (1-\alpha)\boldsymbol{d})^t \boldsymbol{x}' < (\alpha \boldsymbol{c} + (1-\alpha)\boldsymbol{d})^t \boldsymbol{x}^*$. This means that we can find efficient extreme solution by solving the following weighted sum problem which corresponds to a classic shortest path problem.

$$\begin{aligned} \min \quad & (\alpha \hat{\boldsymbol{c}} + (1-\alpha)\boldsymbol{d})^t \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

Hence, the robust solution can be found by computing all efficient extreme solutions of the bicriteria problem. Unfortunately, there is no polynomial bound for the number of efficient extreme solutions of a bicriteria shortest path problem. In fact, it has been shown

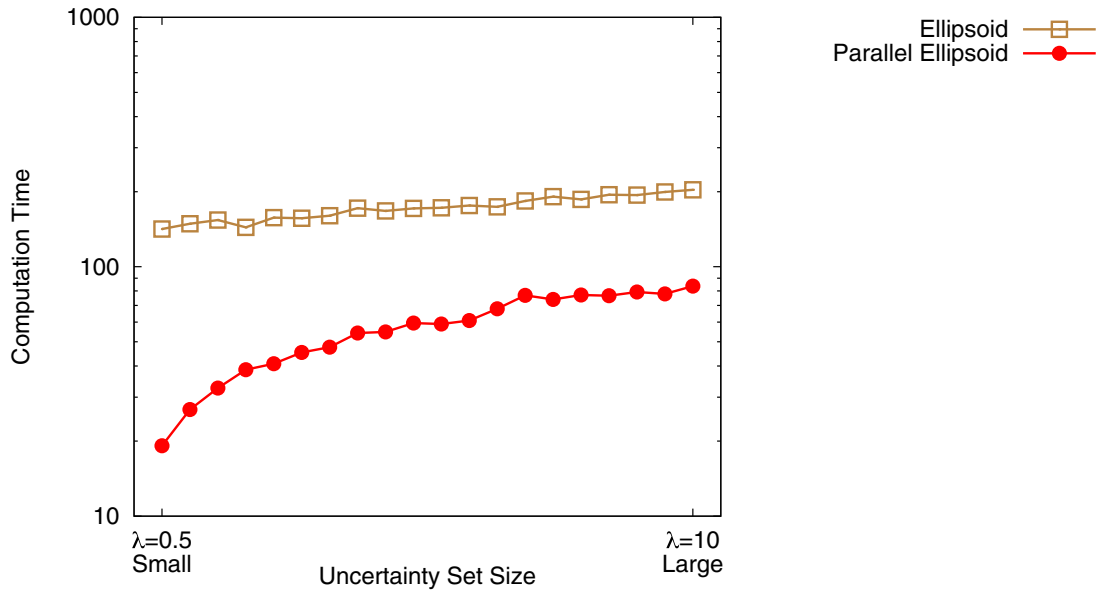


Fig. 7. Comparison between general and axis-parallel ellipsoids, computation times in seconds.

in Carstensen (1983) that there exist instances of the bicriteria shortest path problem with a subexponential number of efficient extreme solutions. Hence, Nikolova (2009) proposes a heuristic to compute only a subset of all efficient extreme solutions of the bicriteria problem. Among the found solutions, the best is chosen with respect to the robust objective function.

In the following, we present an exact algorithm which is guaranteed to find an efficient extreme solution which is optimal for the robust problem without computing all efficient extreme solutions. Unfortunately, we cannot prove that the number of computed solutions by the exact algorithm is polynomially bounded. However, for real-world or randomly generated instances the number of computed solutions is so small that it can be assumed to be constant. We verify this claim in computational experiments. For convenience, we first present a naive algorithm to compute the complete set of all extreme efficient solutions.

4.2.2. Naive algorithm

Note that the lexmin in Step 1 (or Step 2, respectively) of Algorithm 1 can be found by solving a problem of the form

Algorithm 1 Naive algorithm to compute all efficient extreme solutions.

1: Compute $\boldsymbol{x}_l = \text{arglexmin}_{\boldsymbol{x}\in\mathcal{X}}(\hat{\boldsymbol{c}}^t\boldsymbol{x}, \boldsymbol{d}^t\boldsymbol{x})$.
2: Compute $\boldsymbol{x}_r = \text{arglexmin}_{\boldsymbol{x}\in\mathcal{X}}(\boldsymbol{d}^t\boldsymbol{x}, \hat{\boldsymbol{c}}^t\boldsymbol{x})$.
3: **return** EXPLORE$(\boldsymbol{x}_l, \boldsymbol{x}_r)$

$\min_{\boldsymbol{x}\in\mathcal{X}}((1-\epsilon)\hat{\boldsymbol{c}} + \epsilon\boldsymbol{d})^t\boldsymbol{x}$ for a sufficiently small chosen ϵ. The subroutine described as Algorithm 2 recursively finds all efficient

Algorithm 2 Recursive subroutine for the naive algorithm.

1: **procedure** EXPLORE$(\boldsymbol{x}_0, \boldsymbol{x}_1)$
2: Set α_m such that $(\alpha_m\hat{\boldsymbol{c}} + (1-\alpha_m)\boldsymbol{d})^t\boldsymbol{x}_0 = (\alpha_m\hat{\boldsymbol{c}} + (1-\alpha_m)\boldsymbol{d})^t\boldsymbol{x}_1$.
3: Compute $\boldsymbol{x}^* = \text{argmin}_{\boldsymbol{x}\in\mathcal{X}}(\alpha_m\hat{\boldsymbol{c}} + (1-\alpha_m)\boldsymbol{d})^t\boldsymbol{x}$
4: **if** $(\alpha_m\hat{\boldsymbol{c}} + (1-\alpha_m)\boldsymbol{d})^t\boldsymbol{x}^* < (\alpha_m\hat{\boldsymbol{c}} + (1-\alpha_m)\boldsymbol{d})^t\boldsymbol{x}_0$ **then**
5: **return** EXPLORE$(\boldsymbol{x}_0, \boldsymbol{x}^*) \cup \{\boldsymbol{x}^*\} \cup$ EXPLORE$(\boldsymbol{x}^*, \boldsymbol{x}_1)$.
6: **else**
7: **return** $\emptyset$

extreme solutions. The recursion halts if the found solution is not efficient extreme anymore.

We remark that for each efficient extreme solution $\boldsymbol{x}^*$, it is guaranteed that Algorithm 1 either finds $\boldsymbol{x}^*$ or an alternative solution $\boldsymbol{x}_{\text{alt}}$ with $\hat{\boldsymbol{c}}^t\boldsymbol{x}_{\text{alt}} = \hat{\boldsymbol{c}}^t\boldsymbol{x}^*$ and $\boldsymbol{d}^t\boldsymbol{x}_{\text{alt}} = \boldsymbol{d}^t\boldsymbol{x}^*$. Further, it is not guaranteed that all solutions returned by Algorithm 1 are efficient extreme, as the indifference curves $(\hat{c}^t\boldsymbol{x}, \sqrt{\boldsymbol{d}^t\boldsymbol{x}})$ in the objective space are parabolas. However, all non efficient extreme solutions could be easily removed.

4.2.3. Improved algorithm

For the improved algorithm, we first find the two lexicographic minimal solutions $\boldsymbol{x}_l$ and $\boldsymbol{x}_r$ as in the naive method (see Fig. 8(a)). We denote by $p : \mathcal{X} \to \mathbb{R}^2, p(\boldsymbol{x}) = (p_1(\boldsymbol{x}), p_2(\boldsymbol{x})) = (\hat{\boldsymbol{c}}^t\boldsymbol{x}, \boldsymbol{d}^t\boldsymbol{x})$ the map from the solution space to the two-dimensional objective space of the bicriteria optimization problem.

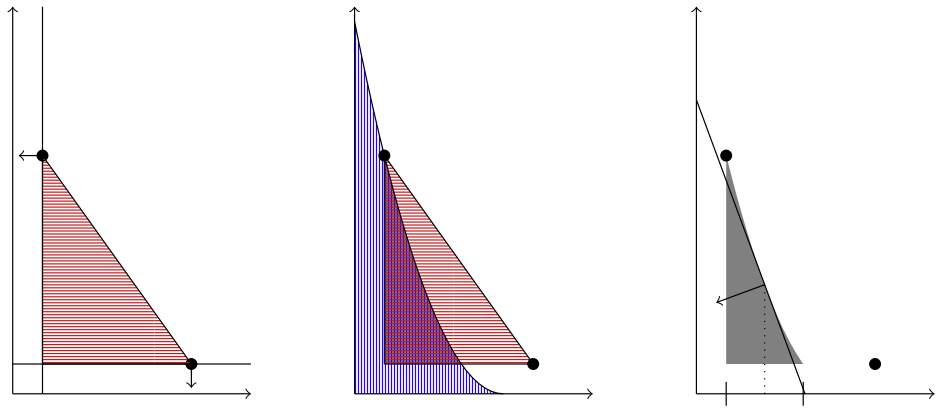
From the definition of efficient extreme solutions, it follows that for each efficient extreme solution $\boldsymbol{x}^*$, $p(\boldsymbol{x}^*)$ is contained in the triangle Δ_{unexp} with the vertices $p(\boldsymbol{x}_l), (p_1(\boldsymbol{x}_l), p_2(\boldsymbol{x}_r))$, and $p(\boldsymbol{x}_r)$ (see Fig. 8(a)).

Denote by $\boldsymbol{x}^*_{\text{rob}} = \text{argmin}(\hat{\boldsymbol{c}}^t\boldsymbol{x}_l + \sqrt{\boldsymbol{d}^t\boldsymbol{x}_l}, \hat{\boldsymbol{c}}^t\boldsymbol{x}_r + \sqrt{\boldsymbol{d}^t\boldsymbol{x}_r})$ the current best solution for the robust problem and by *OBJ* its corresponding objective value. Assume to have a solution $\boldsymbol{x}$ which improves the current best solution. It must hold that

$$\begin{aligned}\hat{\boldsymbol{c}}^t\boldsymbol{x} + \sqrt{\boldsymbol{d}^t\boldsymbol{x}} &< \hat{\boldsymbol{c}}^t\boldsymbol{x}^*_{\text{rob}} + \sqrt{\boldsymbol{d}^t\boldsymbol{x}^*_{\text{rob}}} \\ &\Leftrightarrow \boldsymbol{d}^t\boldsymbol{x} < \left(\hat{\boldsymbol{c}}^t\boldsymbol{x}^*_{\text{rob}} + \sqrt{\boldsymbol{d}^t\boldsymbol{x}^*_{\text{rob}}} - \hat{\boldsymbol{c}}^t\boldsymbol{x}\right)^2 \\ &\Leftrightarrow p_2(\boldsymbol{x}) < (OBJ - p_1(\boldsymbol{x}))^2\end{aligned}$$

This means $p(\boldsymbol{x})$ must be contained in the region $\mathcal{R}_{\text{imp}} = \{z \in \mathbb{R}^2 \mid z_2 < (OBJ - z_1)^2\}$ (see Fig. 8(b)).

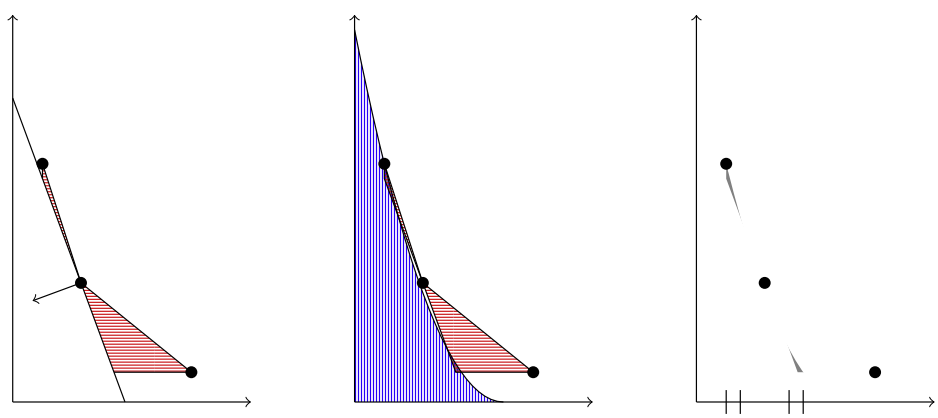
Intuitively, we always have two regions in which we are interested during the algorithm. First, the unexplored region, which may contain efficient extreme solutions which we have not found yet. At the beginning this region corresponds to Δ_{unexp}. Second, the improving region, corresponding to $\mathcal{R}_{\text{imp}}$, which could contain solutions that improve our current best solution. We call the intersection of these two regions $\mathcal{R}_{int}$ in the following. During the algorithm we keep track of $\mathcal{R}_{int}$. The algorithm then solves, for different values of α, shortest path problems with the objective



(a) To initialize the algorithm we compute the two solutions which minimize the first and second objective function. The unexplored region which might contain efficient extreme solutions is marked with horizontal lines.

(b) The region of improvement which might contain solutions that improve the actual best solution is marked with vertical lines. The intersection of both regions, shown in gray, defines the area in which we try to find solutions.

(c) Projecting the gray area to the first axis defines the first interval the algorithm tries to shrink. The algorithm computes the midpoint of the interval and projects it to the parabola to obtain the next direction of optimization



(d) A new solution is found. This solution intersects the previous unexplored region with one half-space, which results in two smaller triangles.

(e) Intersecting the region of improvement with the new unexplored region leads to two small areas.

(f) Projecting the two gray areas to the first axis defines two smaller intervals. The algorithm proceeds by shrinking, splitting or removing these intervals until the gray area corresponds to the empty set

Fig. 8. Visualization of the improved algorithm.

function $\alpha \cdot \boldsymbol{c}^t\boldsymbol{x} + (1-\alpha)\cdot \boldsymbol{d}^t\boldsymbol{x}$. After each solution of such a problem, $\mathcal{R}_{int}$ can be shrunk. If $\mathcal{R}_{int}$ is empty, we have proven to have an optimal solution.

Next we have to describe how we generate from $\mathcal{R}_{int}$ the next α for which we solve the weighted sum problem.

At the beginning, we intersect the triangle Δ_{unexp} and $\mathcal{R}_{\text{imp}}$ and project to the first axis. This results in the interval which we use to initialize a list of intervals $\mathcal{L}$ (see Fig. 8(c)). The idea of the improved algorithm is then to shrink, split or remove intervals from $\mathcal{L}$ until $\mathcal{L}$ is empty.

To do so, the improved algorithm picks an interval $I = [a, b]$ from $\mathcal{L}$, computes its midpoint $m = 0.5(a+b)$ and defines a local approximation of the boundary of $\mathcal{R}_{\text{imp}}$ at $(m, (OBJ - m)^2)$ (see Fig. 8(c)). The slope of the obtained line l is $2(m - OBJ)$. We set $\alpha_m = 2(m - OBJ)/(2(m - OBJ) - 1)$ to optimize in the direction which is perpendicular to l (see Fig. 8(c)).

We then compute $\boldsymbol{x}_{\text{new}} = \text{argmin}_{\boldsymbol{x}\in\mathcal{X}}(\alpha_m\hat{\boldsymbol{c}} + (1-\alpha_m)\boldsymbol{d})^t\boldsymbol{x}$. We know for each other solution $\boldsymbol{x}$ that

$$\begin{aligned}
&\alpha_m\hat{\boldsymbol{c}}^t\boldsymbol{x}_{\text{new}} + (1-\alpha_m)\boldsymbol{d}^t\boldsymbol{x}_{\text{new}} \\
&\quad \le \alpha_m\hat{\boldsymbol{c}}^t\boldsymbol{x} + (1-\alpha_m)\boldsymbol{d}^t\boldsymbol{x} \\
&\quad \Leftrightarrow \frac{\alpha_m}{1-\alpha_m}(\hat{\boldsymbol{c}}^t\boldsymbol{x}_{\text{new}} - \hat{\boldsymbol{c}}\boldsymbol{x}) + \boldsymbol{d}^t\boldsymbol{x}_{\text{new}} \le \boldsymbol{d}^t\boldsymbol{x} \\
&\quad \Leftrightarrow \frac{\frac{2(m-OBJ)}{2(m-OBJ)-1}}{1-\frac{2(m-OBJ)}{2(m-OBJ)-1}}(\hat{\boldsymbol{c}}^t\boldsymbol{x}_{\text{new}} - \hat{\boldsymbol{c}}^t\boldsymbol{x}) + \boldsymbol{d}^t\boldsymbol{x}_{\text{new}} \le \boldsymbol{d}^t\boldsymbol{x} \\
&\quad \Leftrightarrow -2(m-OBJ)(\hat{\boldsymbol{c}}^t\boldsymbol{x}_{\text{new}} - \hat{\boldsymbol{c}}^t\boldsymbol{x}) + \boldsymbol{d}^t\boldsymbol{x}_{\text{new}} \le \boldsymbol{d}^t\boldsymbol{x} \\
&\quad \Leftrightarrow 2(m-OBJ)p_1(\boldsymbol{x}) - 2(m-OBJ)\hat{\boldsymbol{c}}^t\boldsymbol{x}_{\text{new}} + \boldsymbol{d}^t\boldsymbol{x}_{\text{new}} \le p_2(\boldsymbol{x})
\end{aligned}$$

Hence, we conclude that for all efficient extreme solutions $\boldsymbol{x}^*$, $p(\boldsymbol{x}^*)$ must lie above the line through $p(\boldsymbol{x}_{\text{new}})$ with slope $2(m - OBJ)$. Hence, we can exclude a half space from the unexplored region

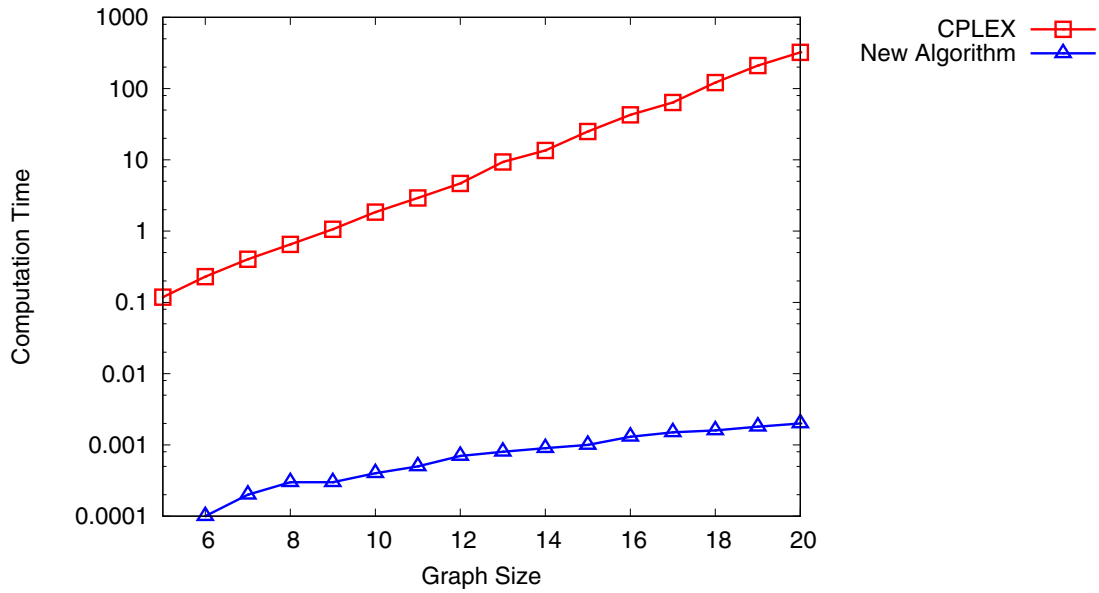


Fig. 9. Comparison of average computation times for grid graphs.

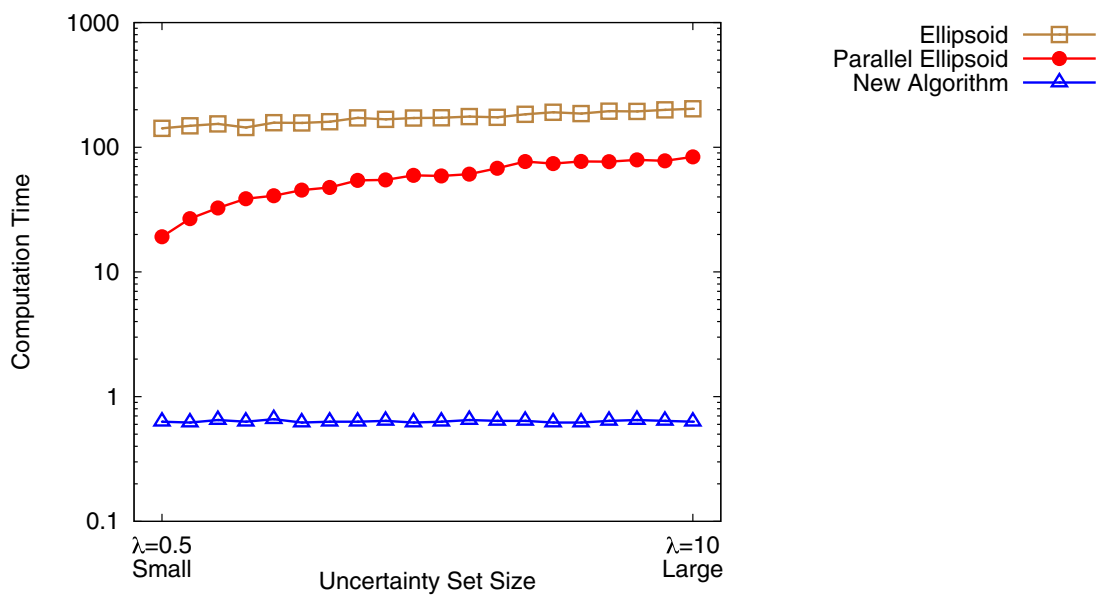


Fig. 10. Comparison between general and axis-parallel ellipsoids, computation times in seconds.

(see Fig. 8(d)) and shrink, split or remove intervals contained in $\mathcal{L}$ (see Fig. 8(e)). Further, it might happen that $\boldsymbol{x}_{\text{new}}$ improves the actual best solution in this case we update *OBJ* and $\mathcal{R}_{\text{imp}}$ and consequently all intervals in $\mathcal{L}$.

If the algorithm has reduced $\mathcal{L}$ to the empty set the current best solution is indeed the optimal solution of the robust optimization problem.

4.2.4. Computational experiments for the improved algorithm

We test the performance of the branch-and-bound algorithm on grid graphs, using the same computational environment as in Section 3. We used the LEMON graph library v.1.3.1 (Dezső, Jüttner, & Kovács, 2011) to solve the classic shortest path problems that needs to be solved during the branch-and-bound algorithm. The goal is to find a path from the upper left corner to the lower right corner of the grid. For each arc we chose 50 values uniform at random from $\{1, \ldots, 100\}$. Further, we fit an axis-parallel ellipsoidal uncertainty set to these points as described previously. We vary the grid size from a 5×5 grid to a 20×20 grid. For each grid size we create 100 instances and solve them in two ways: By using CPLEX to solve the resulting MISOCP and by the proposed branch-and-bound algorithm. The averaged computation times are shown in Fig. 9.

It can be seen that our approach outperforms CPLEX by several orders of magnitude (note the logarithmic vertical scale). While the computation times for CPLEX scale exponentially with the graph size, this is not observed for our method. We show the average number of shortest path calculations required by our method in Table 2, where we increase the grid size to 100×100. It can be seen that on average only very few calls are required, and the increase is slow. By using further improved algorithms for shortest path calculation in road networks (see Bast et al., 2016), the application of our method in real-time route planning is within reach.

Finally, we revisit the computation times for the real-world instance from our previous experiment (Fig. 7). Fig. 10 shows the performance of our method for comparison.

Table 2
Average number of shortest path computations for different grid sizes.

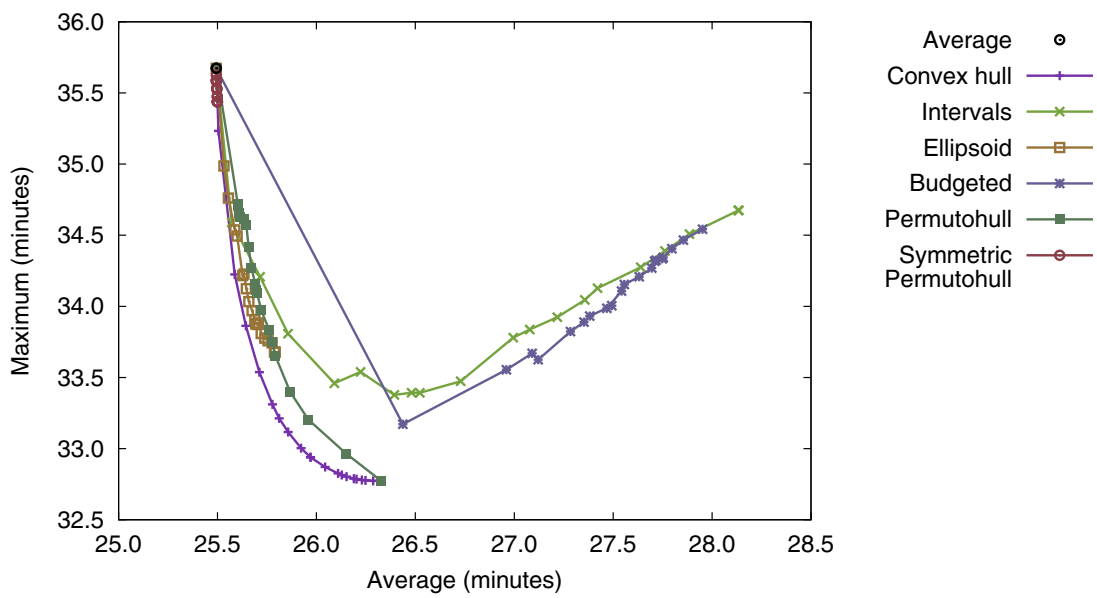
Instance size	SP comp.
5×5	3.625
10×10	3.929
20×20	4.251
50×50	4.778
100×100	5.127

5. Conclusions

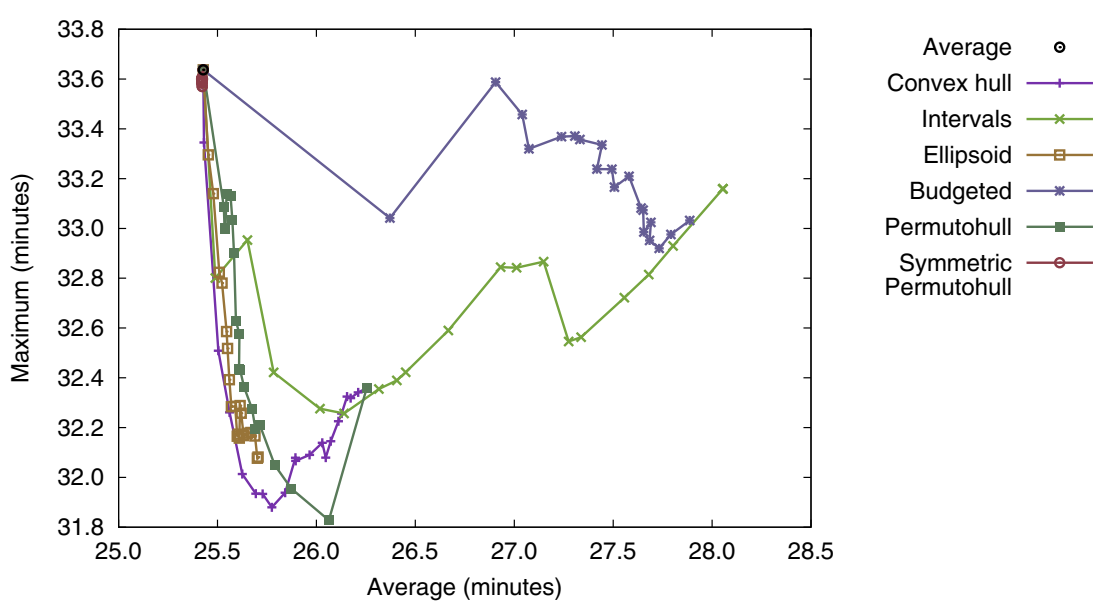
In this paper, we constructed uncertainty sets for the robust shortest path problem using real-world traffic observations for the City of Chicago. We evaluated the model suitability of these sets by finding the resulting robust paths, and comparing their in-sample and out-sample performance using different performance indicators. Naturally, conclusions can only be drawn within the reach of the available data. It remains to be seen how the considered uncertainty sets perform on other datasets for robust shortest paths.

We have observed that using ellipsoidal uncertainty sets provides high-quality solutions with less computational effort than for the permutohull. If one uses only the diagonal entries of the matrix $\mathbf{\Sigma}$, then one ignores the data correlation in the network, but the solution quality remains roughly the same. For the resulting problem, a specialized branch-and-bound algorithm was developed that is able to reduce computation times considerably compared to CPLEX. In fact, the computational effort to solve this problem is comparable to the complexity of solving a few classic shortest path problems, which even makes the application on real-time navigation devices a possibility.

Appendix A. Additional experimental results

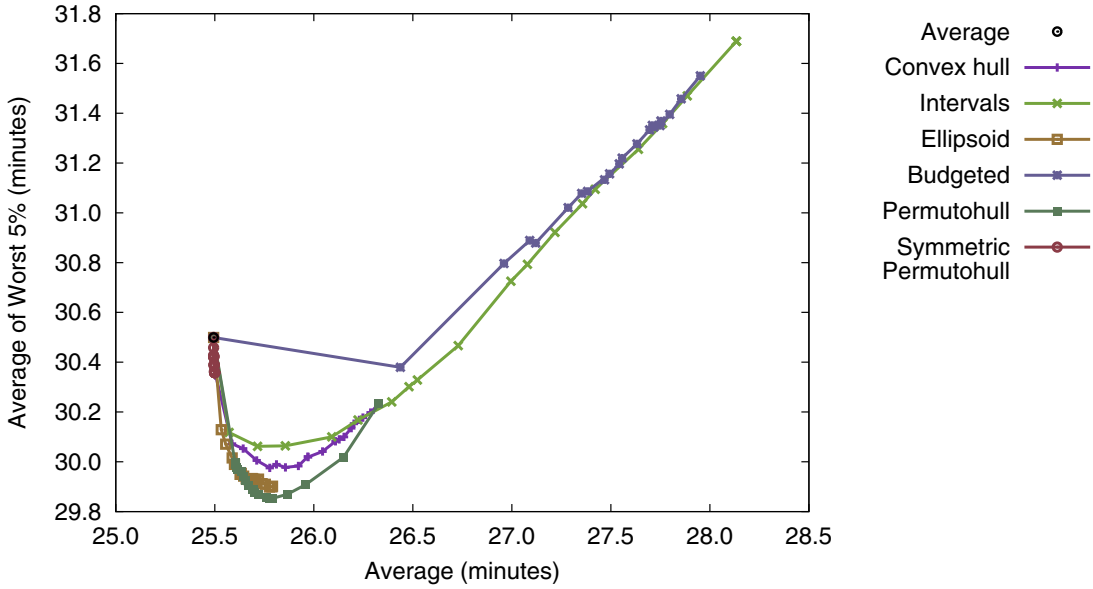


(a) in-sample

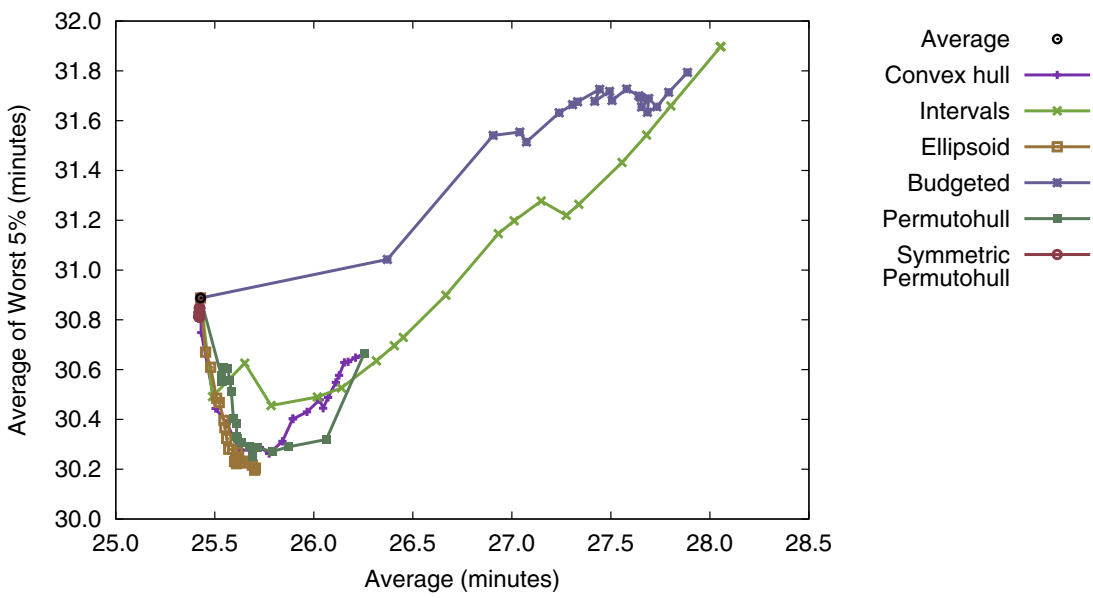


(b) out-sample

Fig. A1. Average vs. worst-case performance, mornings dataset.

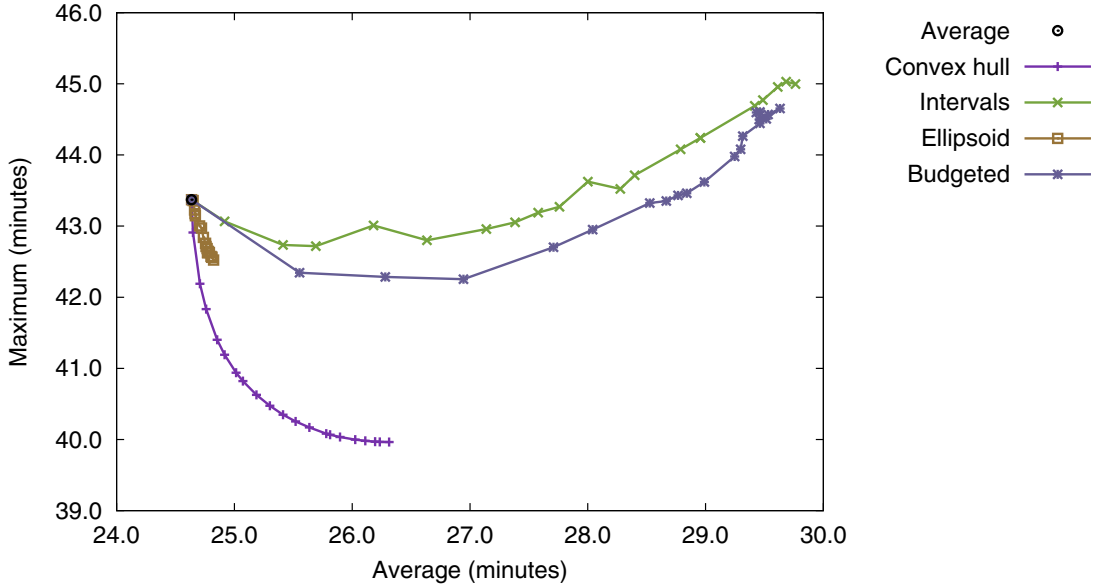


(a) in-sample

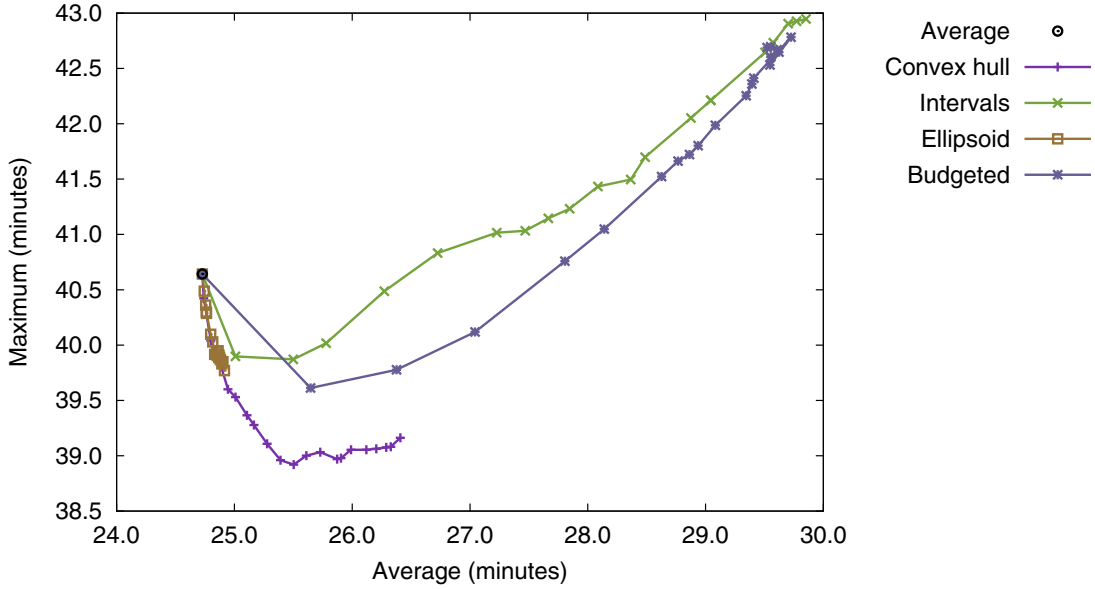


(b) out-sample

Fig. A2. Average vs. CVaR performance, mornings dataset.

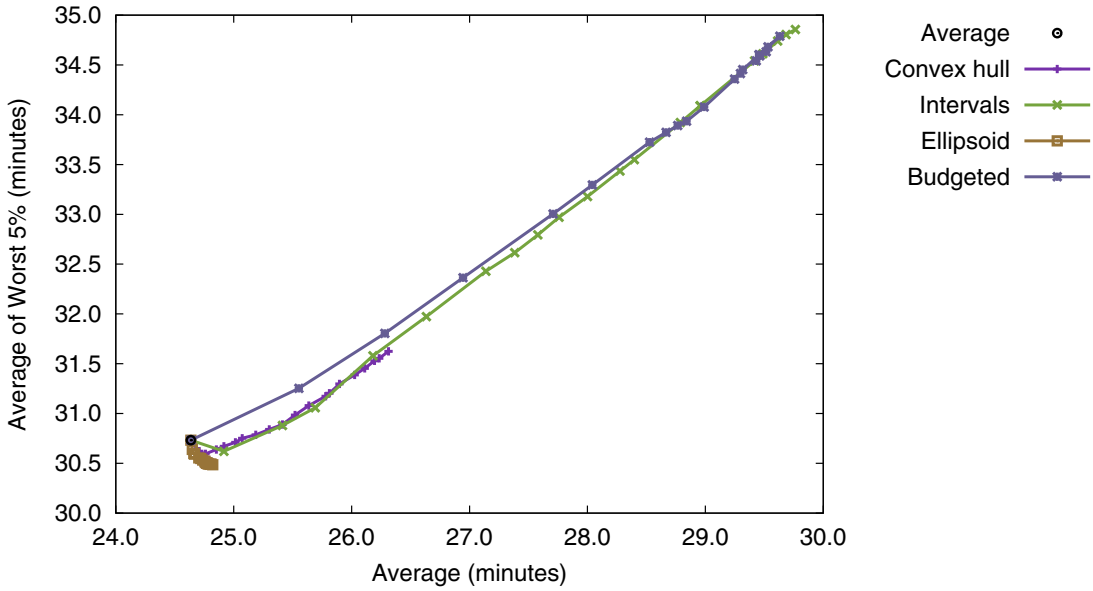


(a) in-sample

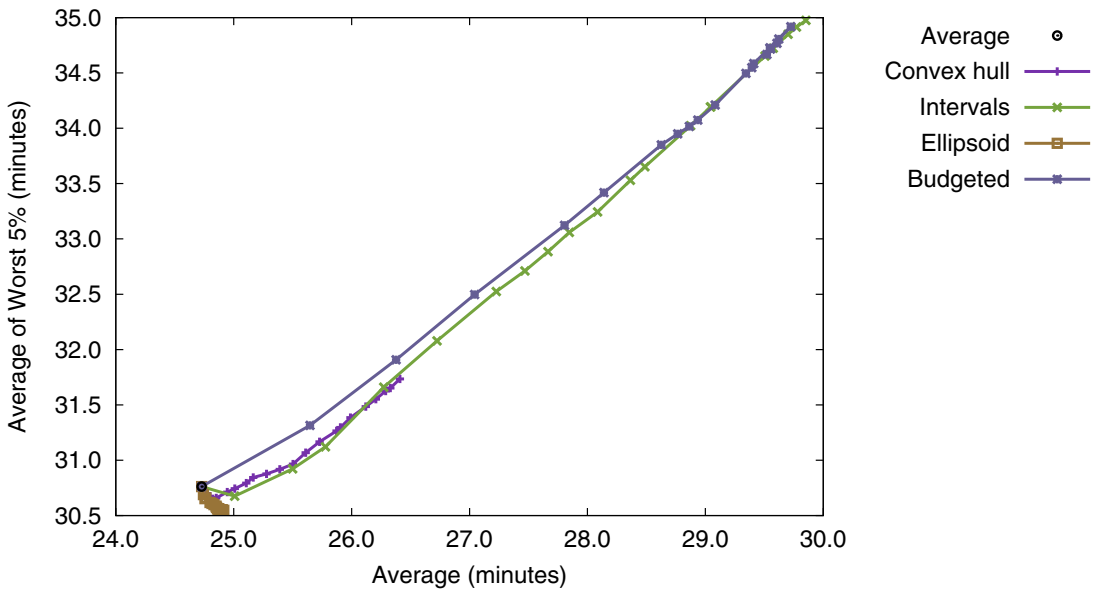


(b) out-sample

Fig. A3. Average vs. worst-case performance, complete dataset.



(a) in-sample



(b) out-sample

Fig. A4. Average vs. CVaR performance, complete dataset.

References

Aissi, H., Bazgan, C., & Vanderpooten, D. (2009). Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research, 197*(2), 427–438.

Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., et al. (2016). Route planning in transportation networks. In *Algorithm engineering* (pp. 19–80). Springer.

Ben-Tal, A., & Nemirovski, A. (1998). Robust convex optimization. *Mathematics of Operations Research, 23*(4), 769–805.

Ben-Tal, A., & Nemirovski, A. (1999). Robust solutions of uncertain linear programs. *Operations Research Letters, 25*(1), 1–13.

Bertsimas, D., & Brown, D. B. (2009). Constructing uncertainty sets for robust linear optimization. *Operations Research, 57*(6), 1483–1495.

Bertsimas, D., Gupta, V., & Kallus, N. (2018). Data-driven robust optimization. *Mathematical Programming, 167.2*, 235–292.

Bertsimas, D., & Sim, M. (2003). Robust discrete optimization and network flows. *Mathematical Programming, 98*(1), 49–71.

Bertsimas, D., & Sim, M. (2004). The price of robustness. *Operations Research, 52*(1), 35–53.

Büsing, C. (2012). Recoverable robust shortest path problems. *Networks, 59*(1), 181–189.

Carstensen, P. J. (1983). *The complexity of some problems in parametric linear and combinatorial programming*. University of Michigan.

Chassein, A., & Goerigk, M. (2015). A new bound for the midpoint solution in min-max regret optimization with an application to the robust shortest path problem. *European Journal of Operational Research, 244*(3), 739–747.

Chassein, A., & Goerigk, M. (2016a). A bicriteria approach to robust optimization. *Computers & Operations Research, 66*, 181–189.

Chassein, A., & Goerigk, M. (2016b). Performance analysis in robust optimization. In *Robustness analysis in decision aiding, optimization, and analytics* (pp. 145–170). Springer.

Chassein, A., & Goerigk, M. (2018). Variable-sized uncertainty and inverse problems in robust optimization. *European Journal of Operational Research, 264.1*, 17–28.

Dezső, B., Jüttner, A., & Kovács, P. (2011). LEMON – an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science, 264*(5), 23–45.

Dokka, T., & Goerigk, M. (2017). An experimental comparison of uncertainty sets for robust shortest path problems. In G. D'Angelo, & T. Dollevoet (Eds.), *Proceedings of the 17th workshop on algorithmic approaches for transportation modelling, optimization, and systems (ATMOS 2017): 59* (pp. 16:1–16:13). Dagstuhl, Germany: OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz–Zentrum fuer Informatik

Goerigk, M., & Schöbel, A. (2016). Algorithm engineering in robust optimization. In *Algorithm engineering* (pp. 245–279). Springer.

IBM ILOG CPLEX (2014). 12.6. CPLEX user's manual.

Kasperski, A., & Zieliński, P. (2016). Robust discrete optimization under discrete and interval uncertainty: A survey. In *Robustness analysis in decision aiding, optimization, and analytics* (pp. 113–143). Springer.

Montemanni, R., & Gambardella, L. M. (2004). An exact algorithm for the robust shortest path problem with interval data. *Computers & Operations Research, 31*(10), 1667–1680.

Nikolova, E. (2009). High-performance heuristics for optimization in stochastic traffic engineering problems. In *Proceedings of the international conference on large-scale scientific computing* (pp. 352–360). Springer.

Yu, G., & Yang, J. (1998). On the robust shortest path problem. *Computers & Operations Research, 25*(6), 457–468.