JULIE MEIßNER

# UNCERTAINTY EXPLORATION

ALGORITHMS

COMPETITIVE ANALYSIS

COMPUTATIONAL EXPERIMENTS

# Uncertainty Exploration

## Algorithms, Competitive Analysis, and Computational Experiments

vorgelegt von
M. Sc. Julie Meißner
geb. in Berlin

von der Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss
Vorsitzender: Prof. Dr. Sullivan
Gutachter: Prof. Dr. Nicole Megow
Prof. Dr. Martin Skutella
Prof. Dr. Leen Stougie

Tag der wissenschaftlichen Aussprache: 7. September 2017

Berlin 2018

Publisher:
Julie Meißner
julie.meissn@gmail.com


Print: epubli – a service of neopubli GmbH, Berlin

# Acknowledgements

This thesis would have not been possible without the support of my family, friends, and colleagues. First and foremost, I thank my supervisors Nicole and Martin for their support and guidance. Suggesting this topic, which immediately caught my interest because of its novelty and relevance, contemplating jointly if 1.707 can be the truth and showing me universities all over Germany, were contributions that set a great framework for my work. Our numerous discussions and joint work as well as your suggestions of related ideas and new concepts facilitated the research for this thesis.

I want to thank José for inviting me for a two month research stay in Santiago de Chile. Our discussions on this and other topics were always inspiring. It has been a pleasure to work with my coauthors Christoph, Jacob, and Thomas and I am thankful to Leen, my colleagues in Chile, and everyone else who expressed interest in the topic and engaged in lively discussions. Thank you to Wiebke, who allowed me to build on her style files, and to Andrea, Max, and Lydia for reading various sections of this document.

The environment at COGA with its unique collegial atmosphere has become a home to me. Thank you to all my former colleagues for many discussions over coffee and during table football matches, fruitful collaborations, and support in many small questions.

I express my thanks to the Deutsche Forschungsgesellschaft and the Einstein Center for Mathematics in the framework of Matheon for funding this research project and the DAAD and Pontificia Universidad Cathólica de Chile for funding my research in Chile.

I wish to thank my family and friends for their never-ending support. Each of you has contributed to this thesis in their own, unique way. My grandfathers mantra ‘wissenschaftliches Arbeiten ist begründetes Arbeiten’ helped with good style, my grandmother’s refuge in Austria brought clear thoughts, or my friends’ activities to enjoy a good time off are just a few examples. Finally, I owe special thanks to Max for his love and unconditional support.

I am extremely grateful to all of you! *Julie Meißner*

# Contents

# Introduction

Combinatorial optimization captures a wide range of challenges such as infrastructure design and scheduling of tasks. For a fixed number of possible new infrastructure connections or tasks to schedule, there are exponentially many infrastructure networks and schedules that can be created from them. This rules out assessing every possible network or schedule for its quality to find the best option as inefficient. Instead, we need more sophisticated, efficient strategies to find a best network or schedule. Due to the many daily life occurrences of combinatorial optimization tasks, they can be tracked far back in history. However, only with the advancement of computers and the development of linear programming as a joint solution tool in the 1950's, they have generated extensive research [Sch05].

A major challenge for the application of combinatorial solutions is data uncertainty. If, for example, the construction cost of a connection within a network or the processing time of a job is only known roughly, this may significantly influence the quality of a solution containing the connection or job. The challenge of optimization with uncertain input data is the topic of three classical research streams. *Robust optimization* [BTEGN09] considers a fixed set of scenarios for the uncertain data. We evaluate each solution by the performance it can attain for all possible scenarios. Then, the evaluation of each solution yields a guarantee on the solution performance, which is completely independent of the scenario that will occur. Naturally, we aim to find a solution with maximal evaluation. In *stochastic optimization* [BL97] the set of scenarios is weighted by a distribution that describes the probability that each scenario occurs. We compare the expected performance of the solutions to find the best one. It performs well on average, but it may perform a lot worse for particular, unlikely scenarios. Both of these areas, robust and stochastic optimization, assume that the set of scenarios is fixed and the occurring scenario is only revealed after the complete solution has been determined. *Online optimization* [BEY98] takes a different approach. Here, there are

no scenarios but instead the input data is revealed sequentially. For example in network design we have to decide for each connection immediately if it is contained in the solution or not; without knowledge of how many and which additional connections will be revealed.

All three approaches treat uncertainty in the input data as a 'set in stone' characteristic of the optimization problem. However, in many contexts it is possible to obtain exact or more precise data at a certain exploration effort. This is exactly the setting we investigate in this thesis. Optimization with *explorable uncertainty* considers combinatorial problems with uncertain input data, where improved or exact data can be explored at an additional cost. A combinatorial problem has many feasible solutions. The quality of a solution depends on the exact input data and thus is unknown when only uncertain input data is available. When investing in more precise data, we query one data point at a time for its exact value. This improves the knowledge about the quality of solutions and consequently allows us to choose a solution of better quality. The goal is to quantify the trade-off between an investment in more precise data and the resulting quality of the solution to the optimization problem.

Such models have received little attention in the research community, even though they regularly occur in practice. A classical application are estimated user demands that can be specified by undertaking a user survey, but this is an investment in terms of time and/or cost. Other applications include insufficient information on existing infrastructure for telecommunication network planning, where a field measurement can reveal the capacity of an existing connection, or scheduling computer programs on a processor whose running time could be improved by an unknown amount by a code optimizer.

A major research line in this context asks for the minimum exploration cost to find an optimal solution for the underlying optimization problem. In a sense, this is the opposite of robust optimization that aims for the best solution with zero exploration cost. Uncertain input data can occur in the objective function, then exploration of uncertain data helps to identify a solution that minimizes or maximizes the objective. A well-studied example of this model is the minimum spanning tree problem with uncertain edge weights.

An alternative is to consider uncertain feasibility. Here, a set of possibly feasible solutions is given and exploring the input data yields additional information about the feasibility. Either, there is a single feasible solution and finding it with little exploration cost is the goal, or there are several feasible solutions. Then, data exploration has to

find the feasible solution that maximizes or minimizes the optimization goal. The $k$-th smallest value problem is a member of the single feasible solution case, while packing a knapsack with uncertain item weights falls in the latter category.

Uncertainty exploration also describes the study of trade-offs between solution quality and exploration cost. Here, one can relax the optimality condition and minimize the exploration cost to find an approximate solution. Or, given a fixed budget for the uncertainty exploration, one aims to maximize the quality guarantee one can give for the solution.

In all of these approaches the exploration cost and the solution quality are assessed separately, like a bi-criteria optimization problem. A new direction combines these two in a single objective function. This occurs when both, exploration and solution, use the same resource – like time, cost, or energy – and thus affect each other. For example in scheduling, computer programs and a code optimizer that could improve the program's running time, both compete for time on the processor.

## Related Work

Uncertainty exploration was probably first studied for the maximum and median value problem [Kah91]. For a set of elements with uncertainty intervals instead of element weights, find the maximum and median by querying a minimum number of elements for their exact weight. These are special cases of the $k$-th smallest value problem, for which an algorithm using at most $OPT + k$ queries exists [GSS16].

Another line of work considers the minimum spanning tree (MST) problem with uncertain edge weights [EHK+08]. There is a best-possible, deterministic 2-competitive algorithm [EHK+08] that can be generalized to the problem of finding a minimum weight basis of a matroid with uncertain weights [EHK16]. The verification problem of finding the optimal query set for a given realization of edge weights can be solved optimally in polynomial time [EH14].

A generalization of this is the cheapest set problem under uncertainty [EHK16]. For a set of elements whose weight is known to lie in an element-individual interval, find the cheapest set from a family of subsets of the element set. There is no constant-competitive algorithm for this setting. However, when the parameter $d$ denotes the largest size of a set in the family, there is an algorithm identifying the cheapest set by querying less than $d \cdot OPT + d$ elements [EHK16] and there cannot be a better algorithm.

Further problems studied in this uncertainty model include computing a function

value [KT01], and combinatorial optimization problems, such as shortest path [FMO+07], finding the median [FMP+03] and minimum multicut in trees [EHK16]. Geometric problems were studied on a set of points with uncertain location in the optimization model [BHKR05] and in the verification version [CH13].

In the offline setting one has to choose a set of queries that ensures to identify an optimal solution, independent of the underlying, unknown realization. This has been studied for caching problems in distributed databases [OW00], shortest path [FMO+07], and finding the median [FMP+03]. These works also initiated the study of trade-offs between the number of queries and the precision of the solution.

A generalized exploration model was proposed in [GSS16], where upon an element query, a refined open or trivial subinterval is revealed and thus multiple queries per edge might be required. They show that the MST algorithm in [EHK+08] can be adapted and still achieves competitive ratio 2. For $k$-th smallest value there is a $2(OPT + k)$-competitive algorithm [GSS16].

For the model where a fixed budget for the exploration cost is given, there are deterministic and randomized algorithms for the knapsack problem with uncertain weights as well as computational experiments [GGI+15].

Research that considers non-uniform query costs has been undertaken on the $k$-th smallest value problem [FMP+03]. They consider the described online and offline versions of the problem and analyze the impact of asking to find the $k$-th smallest element only up to a precision given as the input.

There is also a recent survey of the research on uncertainty exploration [EH15].

The model combining exploration cost and solution quality, which we study in Chapter 5, is inspired by (and draws motivation from) recent work on a stochastic model of scheduling with testing [Lev16, LMS15, Sha16]. They consider minimizing the weighted sum of completion times on one machine for jobs whose processing times and weights are random variables with a joint distribution that are independent and identically distributed across jobs. In their model, testing a job does not make its processing time shorter, it only provides information for the scheduler (by revealing the exact weight and processing time for a job, whereas initially only the distribution is known). They present structural results about optimal policies and efficient optimal or near-optimal solutions based on dynamic programming [LMS15].

## Thesis Outline

In this thesis we investigate the potential, limits, and applicability of optimization with explorable uncertainty. We present new algorithmic results, lower bounds, and computational experiments. In Chapter 1, we study the minimum spanning tree (MST) under uncertainty problem and its extension to matroids using the most popular model for uncertainty exploration. In a given graph, we know initially for each edge only an interval containing the true edge weight. The true value is revealed upon request (we say 'query') at a given cost. The task is to determine a minimum-cost adaptive sequence of queries to find a minimum weight spanning tree. In the basic setting, we only need to guarantee that the obtained spanning tree is minimal and we do not need to compute its actual weight, i. e. , there might be tree edges whose weights we never query, as they appear in an MST independent of their exact weights. We measure the performance of an algorithm by competitive analysis. For any realization of edge weights, we compare the query cost of an algorithm with the optimal query cost. This is the cost for verifying an MST for a given fixed realization. We distinguish between deterministic algorithms and randomized ones. Deterministic algorithms have a predefined behavior for any algorithm state, while randomized ones may decide their behavior randomly according to a fixed probability distribution. This means such an algorithm may choose different behaviors and thus lead to altered results when run repeatedly. We evaluate randomized algorithms by their expected performance.

For minimum spanning tree under uncertainty we develop a randomized algorithm that improves upon the competitive ratio of any deterministic algorithm. This solves an important open problem in this area [EH15]. We also present the first algorithms for non-uniform query costs and generalize the results to matroids with uncertain weights, in both settings matching the best known competitive ratios for MST with uniform query cost. This showcases many characteristics of the topic and raises two important questions. Given the algorithms for MST under uncertainty with small competitive ratio, how is their performance in practice? How far can we extend the problem class beyond MST and matroids and maintain a small competitive ratio?

In Chapter 2 we answer the first question. We conduct the first practical experiments for MST under uncertainty using data from an application in telecommunications and uncertainty instances generated from the standard TSPLib graph library. We consider three algorithms presented in Chapter 1 and compare their empirical behavior to

the theoretical analysis of the algorithms. Among others, we observe that the average performance and the absolute number of queries are both far from the theoretical worst-case bounds. Thus, MST under uncertainty and also matroids are well-understood and can be solved with small exploration cost in theory and practice.

However, we show in Chapter 3 that any generalization increases the competitive ratio. Set systems are given by a ground set of elements together with a family of subsets of the element set. Initially, we know for each element only an interval containing the element weight. The true value is revealed upon a query of the element at a given cost. The task is to determine a minimum-cost adaptive sequence of queries to find a maximum weight set from the family. As before, we can guarantee that the obtained set has maximal weight without computing its actual weight and we measure the performance of an algorithm by competitive analysis. We show that for any set system whose family of inclusion-wise maximal sets does not equal the basis set of a matroid, no algorithm has competitive ratio $c < 3$. Thus, a set system allows an algorithm with competitive ratio 2 if and only if the family of maximal sets equals the basis set of a matroid. We also show non-constant lower bounds for two special cases: matching with uncertain edge weights, even on bipartite graphs, and knapsack with uncertain profits. We consider linear programs with uncertain objective function as a further generalization. Here, we give an alternative, geometric proof of the lower bound.

In Chapter 4 we study facets of uncertainty exploration that diverge from the classical model of minimizing the exploration cost to identify a solution that optimizes the uncertain objective function. The competitive ratio improves for the variant of MST under uncertainty where the weight of the minimum spanning tree has to be computed. When queries may return subintervals instead of points and edges can be queried multiple times, randomization does not improve over deterministic algorithms. At the same time allowing for non-uniform query costs incurs no loss in performance. We also show that identifying an approximate MST under uncertainty does not improve the competitive ratio. This means that the trade-off between solution quality and competitive ratio is constant. Limiting the number of consecutive queries and thus enforcing parallel queries was proposed as an interesting question in [EH15]. We give a lower bound and present a non-trivial algorithm for this model. Additionally, we analyze uncertainty exploration when the feasibility of solutions is uncertain. This includes $k$-th smallest value, sequencing, and knapsack with uncertain weights.

Having seen the possibilities and limits of the classical approach for uncertainty ex-

ploration, we consider a novel model in Chapter 5. We combine the exploration cost and the solution quality in a single objective function for scheduling jobs on a single machine. For a set of jobs, the processing time of a job can potentially be reduced (by an *a priori* unknown amount) by testing the job. Testing a job takes one unit of time and may reduce its processing time from the given upper limit (which is the time taken to execute the job if it is not tested) to any value which is at least 0. The objective is minimizing the sum of completion times. We give a 2-competitive deterministic algorithm and prove a lower bound of 1.85 on the best possible competitive ratio of any deterministic algorithm. This lower bound holds even for instances with uniform upper limits. Furthermore, we show that randomization helps and present a 1.75-competitive randomized algorithm. We also give a lower bound of 1.62 on the best possible competitive ratio of any randomized algorithm. For the special case of uniform upper limits, we give a deterministic algorithm that is 1.93-competitive.

We conclude with a summary of the current research state for uncertainty exploration in combinatorial problems and a discussion of interesting open questions.

## Notation and Formal Problem Description

We introduce our notation for uncertainty exploration in the classical model for the minimum spanning tree problem, which is a reoccurring topic in several chapters. All other problems and models are formally introduced in their respective chapters.

Consider a weighted, undirected, connected graph $G = (V, E)$, with $|V| = n$ and $|E| = m$. Each edge $e \in E$ comes with an *uncertainty interval* $A_e$ and possibly a *query cost* $c_e$. The uncertainty interval $A_e$ constitutes the only information about $e$'s unknown weight $w_e \in A_e$. We assume that an interval with lower limit $L_e$ and upper limit $U_e$ is either *trivial*, i. e. , $A_e = [L_e, U_e], L_e = w_e = U_e$, or it is *open*, $A_e = (L_e, U_e), L_e < U_e$. Closed, non-trivial intervals cannot be allowed as they lead to a non-constant competitive ratio [EHK+08]. We refer to such an instance of our problem as an *uncertainty graph*. A *realization* $\mathcal{R}$ is a set of edge weights $(w_e)_{e \in E}$ where for each edge $e \in E$ its weight $w_e$ lies in the corresponding uncertainty interval, i. e. , $w_e \in A_e$.

The task is to find a minimum spanning tree (MST) in the uncertainty graph $G$ for an a priori unknown, feasible realization $\mathcal{R}$ of edge weights. To that end, we may query any edge $e \in E$ at cost $c_e$ and obtain its exact weight $w_e$ according to $\mathcal{R}$. The goal is to design an algorithm that constructs a sequence of queries that determines an MST at minimum total query cost. For a realization $\mathcal{R}$ of edge weights, a set of queries $Q \subseteq E$ is

*feasible*, if an MST can be determined given the exact edge weights for edges in $Q$ only; that is, given $w_e$ for $e \in Q$, there is a spanning tree which is minimal for any realization of edge weights $w_e \in A_e$ for $e \in E \setminus Q$. We denote this problem as *MST with edge uncertainty* and say *MST under uncertainty* for short. Note that this problem does not necessarily involve computing the actual MST weight.

We evaluate our algorithms by standard competitive analysis. An algorithm is *c-competitive* if, for any realization $(w_e)_{e \in E}$, the solution query cost is at most $c$ times the optimal query cost for this realization. The optimal query cost is the minimum query cost that an offline algorithm (knowing the realization of edge weights) must pay to verify an MST. Note that we do not allow an additive term in the competitive analysis, unless explicitly stated otherwise. The *competitive ratio* of an algorithm Alg is the infimum over all $c$ such that Alg is $c$-competitive. For randomized algorithms we compare the expected query cost to the optimal query cost. Competitive analysis addresses the problem complexity evolving from the uncertainty in the input, possibly neglecting any computational complexity. However, we note that all our algorithms run in polynomial time unless explicitly stated otherwise.

Chapter 1

# Explorable Uncertainty for Minimum Spanning Trees and Matroids

In this chapter, we investigate the minimum spanning tree problem with uncertain edge weights. It is known that there is a deterministic algorithm with best possible competitive ratio 2 (Erlebach et al. [EHK+08]). Our main result is a randomized algorithm with expected competitive ratio $1 + 1/\sqrt{2} \approx 1.7071$, solving the long-standing open problem whether an expected competitive ratio strictly less than 2 can be achieved (Erlebach and Hoffmann [EH15]). Based on structural insights into the problem we design a preprocessing that maximally reduces the data uncertainty and brings the instance into a special form. A relation between the algorithm and online bipartite vertex cover is the key for the analysis of the randomized algorithm.
We also present novel results for non-uniform query cost and matroids.

Remark: The results in this chapter are based on joint work with Nicole Megow and Martin Skutella, published at the *European Symposium on Algorithms 2015* [MMS15] and in *SIAM Journal on Computing* [MMS17]. Parts of Section 1.2 are joint work with Jacob Focke and Nicole Megow and published at the *Symposium on Experimental Algorithms 2017* [FMM17].

The minimum spanning tree (MST) problem with uncertain edge weights is a well-understood example for optimization with explorable uncertainty. A deterministic, best-possible, 2-competitive algorithm was given by Erlebach et al. [EHK+08] and even extended to matroids [EHK16]. The verification problem, finding the minimum number of queries to verify the minimum spanning tree for a fixed realization, can be solved efficiently [EH14]. This uses a connection to the bipartite vertex cover problem. The lower bound construction for deterministic algorithms yields a randomized lower bound of 1.5, which we describe in Section 1.1 and was also noted by [EH15]. This raised the important open question, whether randomization could improve upon the deterministic

algorithms [EH15]. In this chapter, we answer this question affirmatively and consider other questions about MST under uncertainty and its extension to matroids.

We start out with presenting lower bounds and an intuitive example in Section 1.1. The deterministic lower bound on the competitive ratio is 2 [EHK+08] and the randomized lower bound 1.5. Our introductory example gives some intuition to the reader, as it showcases key characteristics of the problem. We continue by presenting a novel preprocessing that reduces the data uncertainty of an instance and characterizing a class of instances which it solves completely in Section 1.2. Using the structural properties gained by the preprocessing, we develop an algorithm framework in Section 1.3 underlying both our randomized algorithm as well as the results for non-uniform query cost. In Section 1.4 we present this randomized algorithm with tight competitive ratio 1.7071, thus beating the best possible competitive ratio 2 of any deterministic algorithm. One key observation is that MST under uncertainty can be interpreted as a generalized online bipartite vertex cover problem. A similar connection for a fixed realization of edge weights was established in [EH14] for the MST verification problem. This allows to borrow and refine ideas from a recent water-filling algorithm for the online bipartite vertex cover problem [WW15].

In Section 1.5 we consider the more general non-uniform query cost model in which each edge has an individual query cost. We observe that this problem can be reformulated within a different uncertainty model, called OP-OP, presented in [GSS16]. The 2-competitive algorithm in [GSS16] is a pseudo-polynomial, 2-competitive algorithm for our problem with non-uniform query cost. We design new, direct and polynomial-time algorithms that are 2-competitive and 1.7071-competitive in expectation using the framework from Section 1.3. To that end, we employ a new strategy carefully balancing the query cost of an edge and the number of cycles it occurs in.

A natural generalization of MST under uncertainty is finding the minimum weight basis of a matroid. Erlebach et al. [EHK16] show a 2-competitive deterministic algorithm for this problem. We generalize our results for randomized algorithms and non-uniform query cost to matroids with uncertain weights, in both settings matching the best known competitive ratios for MST with uniform query cost. We also present two deterministic, 2-competitive algorithms that can be interpreted as the best-in and worst-out greedy algorithm on matroids.

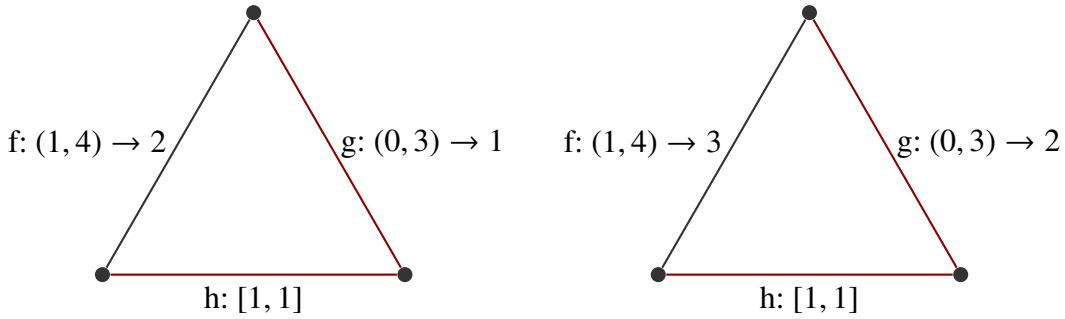



Figure 1.1: Lower bound example with realization $\mathcal{R}_1$ (left) and realization $\mathcal{R}_2$ (right). The edge labels "$e : (L_e, U_e) \to w_e$" give edge $e$'s uncertainty interval $(L_e, U_e)$ as well as its (a priori unknown) weight $w_e$ in a particular realization.

## 1.1 Lower Bounds and Intuition

The basis for understanding the behavior of uncertainty intervals and queries is their interplay on a cycle. This simple graph structure showcases both, lower bound examples and insights about the structure of a feasible query set. Consider a triangle with edge weights such that one edge is in any MST and the other two have overlapping uncertainty intervals (cf. Figure 1.1). We cannot decide which of the two edges is in the MST without querying at least one of them. Any deterministic algorithm decides to query either edge $f$ or edge $g$ first. If it decides to query edge $f$ first, the algorithm has competitive ratio 2 for the realization $\mathcal{R}_1$, where the weight of edge $f$ lies in the uncertainty interval of edge $g$. As the weight of edge $g$ is not in the uncertainty interval of edge $f$, the optimal query set is $\{g\}$. Symmetrically the realization $\mathcal{R}_2$ reveals competitive ratio 2 for all algorithms that query edge $g$ first. Thus, as was already observed in [EHK+08], no deterministic algorithm can achieve a competitive ratio smaller than 2.

Next we consider randomized algorithms for the instance given in Figure 1.1. Each algorithm queries edge $f$ with a certain probability first. We compute the expected competitive ratio for the two realizations $\mathcal{R}_1, \mathcal{R}_2$ parametrized by this probability. It is easy to observe that the best randomized algorithm queries both edges with probability $1/2$ and has expected competitive ratio 1.5. This surprisingly easy example yields the best known lower bound on the competitive ratio for randomized algorithms. This lower bound was independently observed by Erlebach and Hoffmann [EH15].

Theorem 1.1 *For minimum spanning tree under uncertainty, no deterministic algorithm can achieve a competitive ratio smaller than* 2 *and no randomized algorithm can*

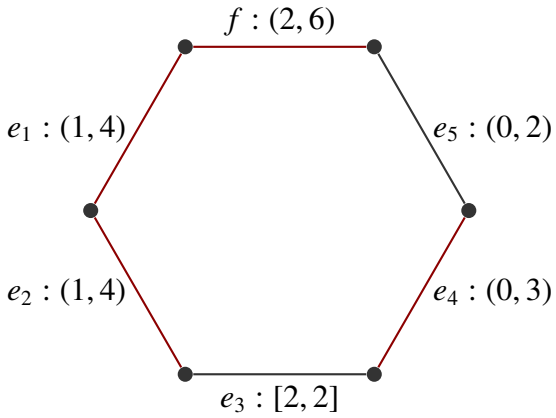



Figure 1.2: Cycle with edge $f$ and edges $e_1, e_2, e_4$ as additional candidates for being maximal.

*achieve a competitive ratio smaller than* 1.5.

We observe important problem features when considering a more general cycle (cf. Figure 1.2). To verify an MST on a cycle, we only need to identify an edge of maximal weight. That means it has largest upper limit on the cycle and either it has a trivial uncertainty interval or no other upper limit exceeds its lower limit. Then there is an MST that does not contain this edge and we can delete it [KV12]. We call such an edge *maximal*. If there is no maximal edge in a cycle, an edge $f$ with the largest upper limit $U_f$ is a natural candidate for being maximal. We first observe that for each such edge $f$, unless we query it, we cannot prove it is contained in an MST, as it has the largest upper limit.

Observation 1.2 *Given a cycle $C$, where no edge is known to be maximal, let $f$ be some edge with largest upper limit $U_f$. Let $\mathcal{R}$ be a realization of edge weights, for which $f$ is in an MST, then $f$ is contained in any feasible query set for $\mathcal{R}$.*

We furthermore observe two different possibilities for proving that an edge $f$ with largest upper limit $U_f$ is in no MST. If edge $f$ has the unique largest lower limit, the only other edges that are candidates for being maximal are the ones whose uncertainty interval overlaps with that of edge $f$. In Figure 1.2 these are the edges $e_1, e_2, e_4$. To find a maximal edge in the cycle we can query edge $f$ to prove its edge weight is larger than the upper limit of all other edges. Instead we can also query all edges with overlapping uncertainty interval and show their edge weight does not exceed $f$'s lower limit. If edge $f$ does not have the unique largest lower limit, the latter option is not feasi-

ble. Thus $f$ must be in any feasible query set. This observation is strengthened and generalized in Lemma 1.13.

Observation 1.3 *Let $f$ be some edge with largest upper limit $U_f$ on a cycle $C$ which does not have a maximal edge.*

1. *For any realization $\mathcal{R}$, every feasible query set contains edge $f$ or all edges in $C$ whose uncertainty interval overlaps that of edge $f$.*

2. *Unless edge $f$ has the unique largest lower limit $L_f$ in $C$, it is in every feasible query set for any realization $\mathcal{R}$.*

## 1.2 Preprocessing

We aim to design an algorithm that starts out with a minimum spanning tree for the particular realization where the weight of each edge is set to its lower limit. All other edges are considered in order of increasing lower limit and the algorithm iteratively tries to add an edge to the current spanning tree, thus closing a cycle. By construction, this cycle is closed by its edge with largest lower limit. The following preprocessing shows that we can modify any instance such that this edge also has the largest upper limit in the cycle. In particular, we can apply Observation 1.3 to this edge. We also show how to adjust the preprocessing such that a maximum number of edges is queried and we describe a particular class of instances that our preprocessing solves completely.

### 1.2.1 Preprocessing Algorithm

Given an uncertainty graph $G = (V, E)$, consider the following two MSTs for extreme realizations. The *lower limit tree* $T_\ell \subseteq E$ is an MST for the realization $w^\ell$, in which all edge weights of edges with non-trivial uncertainty interval are close to their lower limits, more precisely $w^\ell_e = L_e + \varepsilon$ for infinitesimally small $\varepsilon > 0$. Symmetrically, the *upper limit tree* $T_u \subseteq E$ is an MST when the same edges have weight $w^u_e = U_e - \varepsilon$. Each of these sets of edge weights defines an order relation of the edges, where the relation of edges with identical lower (upper) limit is not specified yet. We extend these order relations to an arbitrary but fixed pair of total orderings denoted by $\prec_\ell$ and $\prec_u$. This choice may affect the lower limit tree $T_\ell$ and the upper limit tree $T_u$ w.r.t. these orders.

We investigate at the end of this subsection how this influences the number of queries made in the preprocessing.

Theorem 1.4 *Given an uncertainty graph with lower and upper limit trees $T_\ell$ and $T_u$, any non-trivial edge $e \in T_\ell \setminus T_u$ is in every feasible query set for any realization.*

*Proof.* Given an uncertainty graph, let $h$ be an edge in $T_\ell \setminus T_u$ with non-trivial uncertainty interval. Assume all edges apart from $h$ have been queried and thus have fixed weight $w_e$. As edge $h$ is in $T_\ell$, we can choose its edge weight such that edge $h$ is in any MST. We set $w_h = L_h + \varepsilon$ and choose $\varepsilon > 0$ so small, that all edges with at least the same weight in $w^\ell$ now have a strictly larger edge weight. Symmetrically, if we choose the edge weight $w_h$ sufficiently close to the upper limit $U_h$, no MST contains edge $h$. Consequently we cannot decide whether edge $h$ is in an MST without querying it. □

Our algorithm Preprocessing $(\prec_\ell, \prec_u)$, Algorithm 1.1, applies this theorem repeatedly to a problem instance. We compute the two trees $T_\ell$ and $T_u$ and then query all elements in the set $T_\ell \setminus T_u$ with non-trivial uncertainty interval. This is repeated until this difference contains only edges with trivial uncertainty interval. After at most $m$ repetitions, when all edges have been queried, this definitely is the case and thus the algorithm terminates.

**Algorithm 1.1:** Preprocessing $(\prec_\ell, \prec_u)$

**Input:** An uncertainty graph $G = (V, E)$ and two orderings $\prec_\ell, \prec_u$.

**Output:** A query set $Q \subseteq E$ and the two trees $T_\ell, T_u$.

```
1 Q ← ∅;
2 Determine T_ℓ and T_u according to ≺_ℓ and ≺_u respectively using Prim's
    algorithm [AMO93];
3 while T_ℓ \ T_u contains a non-trivial edge do
4     Query all non-trivial edges in T_ℓ \ T_u, and add them to Q;
5     Update T_ℓ and T_u;
6 Return the query set Q and the two trees T_ℓ, T_u;
```

Corollary 1.5 *Algorithm 1.1 terminates and queries only edges that occur in any feasible query set.*

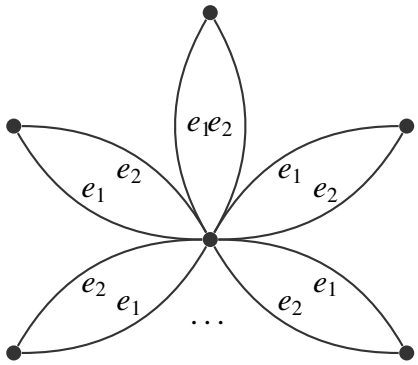



Figure 1.3: Example displaying the importance of the ordering in Preprocessing($\prec_\ell, \prec_u$).

Preprocessing aims at simplifying the input instance, that is, we identify and query edges that must be queried by any algorithm including the optimal one. Naturally, we want to query as many such edges as possible before starting the actual algorithm.

The order relation of edges with identical lower limit or identical upper limit remains unspecified in the definition of $\prec_\ell, \prec_u$ above. Thus, the upper (lower) limit tree is not unique and raises the potential for good or bad choices. As an example, consider a graph of $k$ identical two-edge cycles that are all joined in one node (cf. Figure 1.3). Each cycle is of the form $C = \{e_1, e_2\}$, where all edges have the same lower limit $L$ and upper limits $U_1 < U_2$. Then, for any ordering $\prec_u$ the upper limit tree $T_u$ does not contain $e_2$ for each of the cycles. For the lower limit ordering $\prec_\ell$, all orderings are feasible. For the ordering $e_1 \prec e_2$, we have $T_\ell = T_u$ and the preprocessing does not query any edge. However, for the ordering $e_2 \prec e_1$ the two trees are disjoint and $k$ edges are queried in the first iteration of the preprocessing. Observing this significant impact, we define a specific pair of total orderings $\prec_L, \prec_U$ on the edges and prove that Preprocessing($\prec_L, \prec_U$) maximizes the total number of queries made by the algorithm above.

Definition 1.6 (Limit Orders and Trees) *Let $G = (V, E)$ be an uncertainty graph and let $e_1, \ldots, e_m$ be an arbitrary but fixed labeling of the edges in $E$. Then we define two orderings for the edges in $E$.*

**Lower Limit Order:**

*$e_i \prec_L e_j$, either if $L_{e_i} < L_{e_j}$ or if $L_{e_i} = L_{e_j}$ and one of the following three holds:*

1. *$e_i$ trivial and $e_j$ non-trivial*
2. *$U_{e_i} > U_{e_j}$ and $e_j$ non-trivial*

3. $U_{e_i} = U_{e_j}$ *and* $i < j$.

**Upper Limit Order:**

$e_i \prec_U e_j$, *either if* $U_{e_i} < U_{e_j}$ *or if* $U_{e_i} = U_{e_j}$ *and one of the following three holds:*

1. $e_j$ *trivial and* $e_i$ *non-trivial*
2. $L_{e_i} > L_{e_j}$ *and* $e_i$ *non-trivial*
3. $L_{e_i} = L_{e_j}$ *and* $j < i$.

*We call the corresponding lower and upper limit trees* $T_L$ *and* $T_U$.

We show that Preprocessing$(\prec_L, \prec_U)$ queries all edges which are in $T_\ell \setminus T_u$ for any other pair of orderings $\prec_\ell, \prec_u$. As a first step, it is not hard to see that an edge $e$, which is contained in $T_\ell \setminus T_u$ for some fixed orderings $\prec_\ell$ and $\prec_u$, remains in this set independent from queries of edges other than $e$.

Lemma 1.7 *An edge in* $T_\ell \setminus T_u$ *remains in the set* $T_\ell \setminus T_u$ *until it is queried.*

*Proof.* Let $e$ be in $T_\ell \setminus T_u$. As long as $e$ is not queried, its interval limits do not change. Querying other edges only increases their lower limits and decreases their upper limits. Hence, $e$ stays in $T_\ell$ and remains excluded from $T_u$. □

Next we prove that Preprocessing$(\prec_L, \prec_U)$ does not terminate while there is a non-trivial edge $e$ in the set $T_\ell \setminus T_u$.

Lemma 1.8 *If there is a non-trivial edge in* $T_\ell \setminus T_u$*, then there is also one in* $T_L \setminus T_U$.

*Proof.* Assume there is a non-trivial edge $e \in T_\ell \setminus T_u$, but $T_L \setminus T_U$ contains only trivial edges. We distinguish three cases.

If $e$ is in $T_L$, it is also in $T_U$. Then there is an edge $h$ which is in the cut in $T_U \setminus e$ and in the cycle in $T_u \cup e$. As it is in the cut, we have $U_h \geq U_e$. At the same time, the cycle shows $U_h \leq U_e$, such that the two upper limits must be equal. Then, the fact that $h$ is in the cut, but not in $T_U$ means $L_e \geq L_h$. If $h$ is trivial, $e$ must also be trivial, which contradicts our assumption. Otherwise, as we choose $h \notin T_U$ and $T_L \setminus T_U$ contains only trivial edges, edge $h$ is also not in $T_L$. If $h$ is not in the cut $T_L \setminus e$, there must be an edge $g$ in $T_L \setminus T_U$ that is in the cut $T_U \setminus e$ and in the cycle $T_L \cup h$. This edge $g$ is trivial, larger than $e$ in the ordering $\prec_U$ and smaller than $h$ in the ordering $\prec_L$. This means together with the observations about the bounds of $e$ and $h$ we made above, that we have $U_g \geq U_e = U_h$

and $L_e \geq L_h \geq L_g$. Thus $e$ and $h$ are both trivial: a contradiction. Alternatively, we consider $h$ is in the cut $T_L \setminus e$, where only edges with lower limit at least as large as $L_e$ are contained. This means $L_e \leq L_h$ and consequently $L_e = L_h$. The edge $h$ is in the cut $T_L \setminus e$ and in the cut $T_U \setminus e$, which means we have $e \prec_L h$ and $e \prec_U h$. However, this contradicts that the intervals of $e$ and $h$ are identical.

If $e$ is not in $T_L$ and not in $T_U$, then there is an edge $h$ which is in the cut $T_\ell \setminus e$ and in the cycle in $T_L \cup e$. As it is in the cut, we have $L_e \leq L_h$, and as it is in the cycle we have $L_e \geq L_h$. Thus we have $L_e = L_h$ and $U_h \geq U_e$ because of the ordering of $\prec_L$. We choose $h \in T_L$. As $T_L \setminus T_U$ contains only trivial edges, edge $h$ is also in $T_U$. If $h$ is not in the cycle $T_U \cup e$, there must be an edge $g$ in $T_L \setminus T_U$ that is in the cut $T_U \setminus h$ and in the cycle $T_L \cup e$. This edge $g$ is trivial, larger than $h$ in the ordering $\prec_U$, and smaller than $e$ in the ordering $\prec_L$. This means together with the observations about the bounds of $e$ and $h$ we made above, that we have $U_g \geq U_h \geq U_e$ and $L_h = L_e \geq L_g$. Thus $e$ and $h$ are both trivial: a contradiction. Alternatively we consider $h$ is in the cycle $T_U \cup e$, where only edges with upper limit at most as large as $U_e$ are contained. This means $U_h \leq U_e$ and consequently $U_h = U_e$. Edge $h$ is in the cycle $T_U \cup e$ and in the cycle $T_L \cup e$, which means we have $h \prec_L e$ and $h \prec_U e$. However, this contradicts that the intervals of $e$ and $h$ are identical.

Finally, we consider $e \in T_U \setminus T_L$. Then there is an edge $h$ in the cut $T_U \setminus e$ and in the cycle $T_L \cup e$. This means $h \in T_L \setminus T_U$ and thus edge $h$ is trivial. Additionally, we have $e \prec_U h$ and $h \prec_L e$, which means $L_h \leq L_e \leq U_e \leq U_h$. However, this is a contradiction as $e$ is non-trivial. □

Combined, Lemmas 1.7 and 1.8 yield that Preprocessing$(\prec_L, \prec_U)$ queries every non-trivial edge in $T_\ell \setminus T_u$. This allows us to prove that our preprocessing queries all edges queried in the preprocessing for some other ordering.

**Theorem 1.9** Preprocessing$(\prec_L, \prec_U)$ *queries the union over all edges queried in* Preprocessing$(\prec_\ell, \prec_u)$ *for all orderings* $\prec_\ell, \prec_u$. *Thus, it queries the maximum number of edges characterized by Theorem 1.4.*

*Proof.* We show by induction that edges queried in Preprocessing $(\prec_\ell, \prec_u)$ are also queried in Preprocessing $(\prec_L, \prec_U)$. By Lemmas 1.7 and 1.8, all edges queried in the first iteration of Preprocessing $(\prec_\ell, \prec_u)$ are also queried in our specific preprocessing. Let $e$ be an edge that is queried in iteration $i > 1$ of Preprocessing $(\prec_\ell, \prec_u)$. Let $S$ be the set of all edges queried in the previous iterations. Then, by induction, the set $S$

is queried by Preprocessing $(\prec_L, \prec_U)$. Assume edge $e$ is not queried by Preprocessing $(\prec_L, \prec_U)$. We consider Preprocessing $(\prec_\ell, \prec_u)$ in iteration $i$ and additionally query all edges which are queried by Preprocessing $(\prec_L, \prec_U)$. By Lemma 1.7 edge $e$ is still in $T_\ell \setminus T_u$ for this new uncertainty graph. However, this is exactly the uncertainty graph at the end of Preprocessing $(\prec_L, \prec_U)$. Thus, the termination of the algorithm at this point contradicts Lemma 1.8. □

Preprocessing $(\prec_L, \prec_U)$ queries only edges in $T_L \setminus T_U$ and at the end there are no non-trivial edges in $T_L \setminus T_U$. The existence of non-trivial edges in $T_L \setminus T_U$ increases the size of every feasible query set, in particular also of the optimal query set. Hence, it decreases the competitive ratio of an instance. Thus, when analyzing the worst-case competitive ratio of an algorithm, we can restrict to instances where the preprocessing does not query any edge. As we choose the same ordering for $T_L$ and $T_U$ for trivial edges, this means $T_L = T_U$.

Assumption 1.10 *Without loss of generality we restrict to uncertainty graphs for which $T_L = T_U$ holds.*

### 1.2.2 Instances Solved by the Preprocessing

The preprocessing is a modification of the input instance and intuitively it simplifies it by removing uncertainty. We note, however, that in theory it can lead to a worse algorithm performance for specific input. Nevertheless, in the experiments we present in Chapter 2, the preprocessing generally improves the performance ratio of our algorithms. One class of our data sets is even solved exactly by the preprocessing alone. We generalize this observation and characterize a family of uncertainty graphs which can be completely solved by our preprocessing.

Proposition 1.11 *For uncertainty graphs, in which every cycle contains only edges with identical lower limit or only edges with identical upper limit, the algorithm* Preprocessing$(\prec_L, \prec_U)$ *finds an optimal solution.*

*Proof.* The proof is by contradiction. Assume Preprocessing $(\prec_L, \prec_U)$ terminates with $T_L$ and $T_U$ and did not find a feasible query set. Then the uncertainty graph has a cycle $C$ on which it is unclear which edge has the largest weight. All but one edge of $C$ are in $T_L$. Assume, that originally all edges on the cycle had the same upper limit. If there is only one edge $f$ with largest upper limit, all other edges on the cycle are trivial. Since

it is unclear, which edge has largest weight on $C$, $f$ cannot also have the largest lower limit. Thus $f$ is non-trivial and in $T_L \setminus T_U$, which is a contradiction. Otherwise, there are two non-trivial edges $e$ and $f$ on $C$ with largest upper limit, $e \in T_L$ and $f \notin T_U$. This means we can define an alternative ordering $\prec_u$ with $f \prec_u e$ and thus $e \notin T_u$. Thus, for the preprocessing with orderings $\prec_L$ and $\prec_u$ we have $e \in T_L \setminus T_u$. By Theorem 1.9 this means $\textsc{Preprocessing}(\prec_L, \prec_U)$ queries $e$, a contradiction to $e$ being non-trivial.

A cycle with identical lower limits can be treated analogously. □

## 1.3 A New Algorithm Framework

We design an algorithmic framework for *MST under uncertainty*, which allows to plug in several different algorithmic cores. It is the basis for both, our randomized algorithm and our algorithm for non-uniform query costs. The algorithm Framework is an adaption of the deterministic algorithm for the problem presented in [EHK+08] in the sense that it also relies on the cycle characterization of MSTs: Every edge not in a particular minimum spanning tree is maximal in the cycle it closes when added to the MST.

Given an uncertainty graph $G = (V, E)$ our algorithm Framework starts with a lower limit tree $T_L$. We can view this as a first candidate for an MST we want to verify. We consecutively try to add the other edges $f_1, \ldots, f_{m-n+1} \in R := E \setminus T_L$ to it in order of increasing lower limit; in case of ties we prefer the edge with the smaller upper limit. In every iteration $i = 0, \ldots, m - n + 1$ we maintain a minimum spanning tree verified for the already considered edge set $E_i := T_L \cup \{f_1, \ldots, f_i\}$. That is, we maintain a nested chain of subsets $\emptyset = Q_0 \subseteq Q_1 \subseteq \cdots \subseteq Q_{m-n+1}$ such that $Q_i \subseteq E_i$ is a feasible query set for $E_i$. When we try to add edge $f_i$ to the current spanning tree in iteration $i$, we consider the cycle $C_i$ it closes and query edges until we find a maximal edge on $C_i$. Once we find such an edge, we delete it, as there is an MST not containing this edge. Then we start a new iteration and take the next edge of the sequence into account. A formal description of this procedure is given further below in Algorithm 1.2.

This algorithmic structure allows us to prove two lemmas about any feasible query set and thus, in particular, the optimal feasible query set. The first lemma shows that any feasible query set for the entire uncertainty graph $G = (V, E)$ also verifies a minimum spanning tree for the subgraph $G_i = (V, E_i)$. This crucially relies on the fact that we add edges ordered by increasing lower limit.

Lemma 1.12 *Let $i \in \{0, \dots, m-n+1\}$. Given a feasible query set $Q$ for the uncertainty graph $G = (V, E)$, then the set $Q|_{E_i} := Q \cap E_i$ is a feasible query set for $G_i = (V, E_i)$.*

*Proof.* For some fixed realization of edge weights, let $T$ be a minimum spanning tree of $G$ certified by the feasible query set $Q$. We construct a minimum spanning tree $T'$ of $G_i$ by solely using information provided by the query set $Q|_{E_i}$.

We first argue that there is a minimum spanning tree of $G_i$ that contains every edge in $T \cap E_i$: Consider an edge $e \in T \cap E_i$ and let $U \subset V$ be the subset of nodes in one of the two connected components obtained by deleting $e$ from $T$. Since $Q$ is a feasible query set, it certifies that $e$ has minimal weight among all edges in $E$ connecting $U$ to its complement $V \setminus U$. As a consequence, query set $Q|_{E_i}$ certifies that $e$'s weight is minimal among all edges in $E_i$ connecting $U$ and $V \setminus U$.

We proceed to deal with edges in $E_i \setminus T$ by distinguishing two cases. The first case is that adding edge $e \in E_i \setminus T$ to $T \cap E_i$ closes a cycle $C$. Then, adding edge $e$ to tree $T$ closes the same cycle $C$. Thus, as $Q$ is a feasible query set, it certifies that edge $e$ is maximal on $C$. Moreover, since $C \subseteq E_i$, query set $Q|_{E_i}$ obviously suffices as a certificate. We can therefore discard every such edge $e$.

The second case is, that adding edge $e \in E_i \setminus T$ to the spanning tree $T$ closes a cycle $C$ containing some edge $f \notin E_i$. The feasible query set $Q$ certifies that $e$'s weight is at least as large as the weight of any edge on $C$ including edge $f$. Notice that $L_e \leq L_f$ due to our ordering of edges by increasing lower limit. Thus, in order to certify that $e$'s weight is not smaller than $f$'s weight, its exact weight $w_e$ must be known.

Summarizing, the query set $Q|_{E_i}$ certifies that edges in $T \cap E_i$ can be included into $T'$, certain edges can be safely discarded, and the exact weights of all remaining edges in $E_i$ are known. The gathered information clearly suffices to find a minimum spanning tree $T'$ and, as a consequence, $Q|_{E_i}$ is indeed a feasible query set for $G_i$. □

In the next lemma we give a precise characterization of the edges which a feasible query set contains. This characterization is similar to the so-called 'Witness Set Lemma', that is used in [EHK+08] for their deterministic algorithm.

Lemma 1.13 *For some realization of edge weights, let $T$ be a verified MST of the graph $G_i = (V, E_i)$ and let $C$ be the cycle closed by adding edge $f_{i+1}$ to $T$. Furthermore, let $h$ be some edge with the largest upper limit in $C$ and $g \in C \setminus h$ be an edge with $U_g > L_h$. Then any feasible query set for $G_{i+1} = (V, E_i \cup \{f_{i+1}\})$ contains $h$ or $g$. Moreover, if $A_g$ is contained in $A_h$, any feasible query set contains edge $h$.*

*Proof.* Consider a minimum spanning tree $T'$ for $G_{i+1}$. We distinguish two cases depending on edge $h$ being in the tree $T'$ or not. If $h \in T'$, any feasible query set must identify an edge of larger weight on the cycle $C$. Edge $h$ has the maximal upper limit $U_h$ among all edges in $C$ and thus it must be queried for that purpose. Hence, in this case, edge $h$ is in any feasible query set.

If edge $h$ is not in the tree $T'$, then $h$ must have maximal weight in $C$. In particular, a query set must verify that $h$'s weight is at least as large as $g$'s weight. The uncertainty intervals of these two edges overlap, and thus any feasible query set contains at least one of the two edges. Moreover, if $g$'s uncertainty interval is contained in that of edge $h$, querying $g$ does not reveal any information about the ordering of the two edge weights. Hence, $h$ must be contained in any feasible query set in this case. □

The key to our framework is the preprocessing which results in Assumption 1.10, as it yields structural information for the algorithm.

Lemma 1.14 *For some realization of edge weights, let $T$ be a verified MST of the graph $G_{i-1} = (V, E_{i-1})$ and let $C$ be the cycle closed by adding edge $f_i$ to $T$. Then edge $f_i$ has the largest upper limit in the cycle $C$.*

*Proof.* By Assumption 1.10, $T_L = T_U$ and thus edge $f_i$ is not in the upper limit tree $T_U$. This means, if the tree $T$ has not changed and equals $T_L$, edge $f_i$ has the largest upper limit in the cycle it closes with the tree. If the tree has changed, some edge $e \in T_L$ on it has been replaced by an edge $f_j$, $j < i$, in $T$. This replacement happened when $e$ and $f_j$ where on a cycle $C_j$ and some edges from $C_j$ may now be in $C$. However, as edge $e$ was deleted, $f_j$'s weight, and the weight or upper limit of all other edges on the cycle $C_j$ must be smaller than the upper limit of the deleted edge $e$. Thus, the cycle $C$ closed by edge $f_i$ with $T$ now contains other edges, but the upper limits never increase. □

Thus, when we apply Lemma 1.13 in the analysis of our algorithm, any edge $f_i$ has the role of edge $h$ in the cycle it closes. This means that any feasible query set contains either edge $f_i$ or all edges with uncertainty interval overlapping that of edge $f_i$. Moreover, by Observation 1.2, if edge $f_i$ is in the tree $T_i$ (the MST we verified for $G_i$), then edge $f_i$ must have been queried. Consequently, all edges that are not in the lower limit tree and, in a later iteration, occur as edge $g$ in Lemma 1.13 have already been queried. We can thus restrict to consider those edges as $g$-edges that are in $T_L$. We call them *neighbors* of $f_i$ and let the *neighbor set* $X(f_i)$ contain all edges $e \in C_i \cap T_L$ that have an overlapping uncertainty interval $U_e > L_{f_i}$.

Corollary 1.15 *Given an uncertainty graph $G = (V, E)$ and a realization of edge weights, let $T_L$ be its lower limit tree. Let $T$ be a verified MST of the graph $G_i = (V, E_i)$ and let $C$ be the cycle closed by adding edge $f_{i+1}$ to $T$. Furthermore, let $X(f_{i+1}) \subseteq C \cap T_L$ be the neighbor set. Then any feasible query set contains $f_{i+1}$ or $X(f_{i+1})$.*

Furthermore, Assumption 1.10 yields that after querying edge $f_i$ or the neighbor set, the conditions for the second part of Lemma 1.13 are always fulfilled.

Lemma 1.16 *Given a cycle $C$ on which we have queried edge $f$ with the largest lower limit or all its neighbors $X(f)$ and still no edge is known to be maximal. Then any edge $e \in C$ with largest upper limit on the cycle (which may now be different from edge $f$) is in any feasible query set.*

*Proof.* We distinguish two cases and show for both that we can apply Lemma 1.13: If edge $f$ was queried but is still not known to be maximal, its edge weight lies in the uncertainty interval of $e$, as edge $f$ has the largest lower limit. If all neighbors of $f$ were queried, we have $e = f$. This is because by Assumption 1.10 edge $f$ has the largest upper limit on $C$. Furthermore, the edge weight of one of $f$'s neighbors lies in $A_f$, as $f$ is not maximal. Thus, for both cases edge $e$ is in any feasible query set by Lemma 1.13. □

Hence, we can extend the framework by the following two steps on a cycle $C_i$ without an edge that is known to be maximal. First, we call an algorithm Core which somehow decides between querying edge $f_i$ and its neighbor set $X(f_i)$. If this query does not identify a maximal edge, we continue querying edges in the cycle in order of decreasing upper limit. A formal description of our algorithm is given in Algorithm 1.2.

As pointed out above, the algorithm maintains a verified MST for a subset of the edges of increasing size. At the end of the algorithm the tree is verified for the complete edge set $E$ and thus $Q$ is a feasible query set. Framework terminates, as in each iteration of the while loop an edge is queried. As soon as all edges on a cycle have been queried, we have certainly identified a maximal edge.

Any edge that is queried within Framework outside algorithm Core is in any feasible query set by Lemma 1.16. Thus the competitive ratio of an algorithm is solely determined by the query strategy of algorithm Core.

Remark 1.17 Framework together with the following simple Core procedure yields a variant of the 2-competitive deterministic algorithm by Erlebach et al. [EHK+08]:

**Algorithm 1.2:** FRAMEWORK

**Input:** An uncertainty graph $G = (V, E)$.

**Output:** A feasible query set $Q$.

```
1  Determine a lower limit tree T_L and set the temporary graph Γ to T_L;
2  Index the edges in R := E \ T_L by increasing lower limit f_1, ..., f_{m-n+1};
3  Initialize Q = ∅;
4  for i = 1 to m − n + 1 do
5      Add edge f_i to the temporary graph Γ and let C_i be the unique cycle closed;
6      Let the neighbor set X(f_i) be the set of edges g ∈ T_L ∩ C_i with U_g > L_{f_i};
7      if X(f_i) is not empty then
8          Use algorithm CORE to decide between querying f_i and X(f_i);
9      while no edge in the cycle C_i is known to be maximal do
10         Query an edge e ∈ C_i \ Q with largest upper limit U_e in C_i and add it to
           the query set Q;
11     Delete a maximal edge from Γ;
12 Return the query set Q;
```

Query edge $f_i$ and an arbitrary edge with non-trivial uncertainty interval from $X(f_i)$, if such an edge exists. By Corollary 1.15 at least one of the edges queried in each iteration of CORE is in any feasible query set. The iterations query disjoint edge sets of size at most two, making this a 2-competitive algorithm. It's main advantage is the fact, that is does not need to restart after each query pair.

## Relation to Vertex Cover

It was already observed by Erlebach and Hoffmann [EHK16] that *MST under uncertainty* has a close relation to the vertex cover problem. They show that for a fixed realization we can design a bipartite vertex cover graph using the relation of Lemma 1.13. Then any feasible query set contains a vertex cover of this graph. We generalize the use of this relation to a complete problem instance. We create a bipartite vertex cover graph online along the execution of the FRAMEWORK and thus prove a connection to the online bipartite vertex cover problem.

In the online bipartite vertex cover problem one side of the bipartite graph is given (consisting of the so-called *offline vertices*) and the vertices of the other side appear on-

line one by one together with their incident edges. In any iteration we have to maintain a feasible vertex cover of the revealed graph.

For an instance of *MST under uncertainty* we generate the graph as follows: All edges of the lower limit tree $T_L$ form the offline vertices of the vertex cover graph. During an execution of the algorithm Framework we add the edge $f_i \in R$ to the temporary graph $\Gamma$ such that it closes a unique cycle $C_i$. Upon adding edge $f_i$ in the algorithm, we add a corresponding vertex to the vertex cover graph and connect this new vertex to all vertices corresponding to edges in $C_i \cap T_L$ with overlapping uncertainty interval. Thus the set of neighbors of the new vertex corresponds to the neighbor set $X(f_i)$.

Observe that the vertex cover graph we create depends on the realization of the edge weights. We determine the maximal edge for every cycle $C_i$ and delete it. This determines which cycle is closed next and thus the next incidences in the vertex cover graph. Thus, we need to create the vertex cover graph online and cannot do it a priori.

## 1.4 Randomized Algorithm

In this section we describe a randomized algorithm for *MST under uncertainty* that achieves competitive ratio $1 + 1/\sqrt{2} \approx 1.7071$. Our algorithm Random employs the algorithm Framework presented in the previous section and makes use of its vertex cover interpretation for the algorithm core. We decide how to resolve cycles by maintaining an edge potential for each edge $e \in T_L$. It describes the probability to query an edge. The edge potentials are increased in every cycle we consider throughout the algorithm. To determine the increase, we carefully adapt a water-filling scheme presented in [WW15] for online bipartite vertex cover. This scheme considers all edges queried in Core, but not those queried in Framework. This is the reason that our algorithm does not achieve the same competitive ratio as for online bipartite vertex cover. In this section we assume uniform query cost $c_e = 1$, $e \in E$, and explain the generalization to non-uniform query costs in Section 1.5.

Our algorithm Core of Random is the procedure that decides which edges to query on a cycle $C_i$ in Framework. We maintain an edge potential $y_e \in [0, 1]$ for all edges $e \in T_L$ which is initially set to 0. We query an edge if its potential exceeds the query bound $b$, which we draw uniformly at random from $[0, 1]$ before we start the algorithm Framework. Thus we can interpret the potential as the probability that edge $e$ is queried.

We identify the following goals for the algorithm design: First, edges in the neighbor set of $f_i$ should be queried with high probability, as they can occur in further neighbor sets later. Second, if an edge $e$ in the neighbor set is queried with probability $y_e$, edge $f_i$ must be queried at least with probability $1 - y_e$ to ensure feasibility. And third, in expectation we cannot query more than $1 + \alpha$ edges per iteration to achieve competitive ratio $1 + \alpha$. Here, $\alpha$ is a fixed parameter that is determined later in the analysis. Formally we achieve these goals by distributing no more than potential $\alpha$ among the neighbor set $X(f_i)$. We distribute the potential among all neighbors such that they reach an equal level $t(f_i) \in [0, 1]$ which is as large as possible. This means when we increase $y_e$ to $\max\{t(f_i), y_e\}$ for all neighbors $e \in X(f_i)$, the total potential increase sums up to at most $\alpha$. Now we compare this threshold $t(f_i)$ to the query bound $b$ to decide which edges to query. If $b$ is the larger of the two, we query edge $f_i$ and otherwise we query all neighbors, the edges in $X(f_i)$.

**Algorithm 1.3:** Core of Random

**Input:** A cycle $C_i$ of the algorithm Framework with its edge $f_i$, neighbor set $X(f_i)$, as well as the edge potentials $y_e = y_e^i$ and the query bound $b$.

**Output:** A feasible query set $Q \subseteq C_i$.

1 Maximize the threshold $t(f_i) \leq 1$ s.t. $\sum_{e \in X(f_i)} \max\{0, t(f_i) - y_e\} \leq \alpha$;
2 Increase edge potentials $y_e := \max\{t(f_i), y_e\}$ for all edges $e \in X(f_i)$;
3 **if** $t(f_i) < b$ **then**
4  Add edge $f_i$ to the query set $Q$ and query it;
5 **else**
6  Add all edges in $X(f_i)$ to the query set $Q$ and query them;
7 Return the query set $Q$;

The join of the two algorithms Framework and Core of Random together with the preceding random choice of $b$ and initially setting $y_e := 0$, $e \in T_L$, forms the algorithm Random. This algorithm has competitive ratio $1 + 1/\sqrt{2} \approx 1.7071$ for *MST under uncertainty*, if we choose the parameter $\alpha$ to be $1/\sqrt{2}$.

For the proof of this performance we use an amortized analysis over all cycles closed during the run of the algorithm. We consider a fixed realization of edge weights and a corresponding optimal query set $Q^*$. We denote the potential of an edge $e \in T_L$ at the start of iteration $i$ by $y_e^i$ and use $y_e$ to denote the edge potential after the last iteration

of the algorithm. We will relate the expected number of queries of Random to the total edge potential we distribute. For this, we first bound the potential distributed to edges in $T_L \setminus Q^*$ by the number of edges in $R \cap Q^*$ times our parameter $\alpha$ (where $R = E \setminus T_L$).

Lemma 1.18 *Given an instance of* MST under uncertainty *together with a realization of edge weights, the edge potentials after an execution of* Random*, and any feasible query set $Q^*$, it holds that*

$$\sum_{e \in T_L \setminus Q^*} y_e \leq \alpha \cdot |R \cap Q^*|.$$

*Proof.* For any edge $e \in T_L \setminus Q^*$, Corollary 1.15 states that all neighboring edges $f \in R$ with $e \in X(f)$ must be in the optimal query set $Q^*$. The potential $y_e$ is the sum of the potential increases caused by edges $f \in R$ with $e \in X(f)$. As in each iteration of the algorithm the total increase of potential is bounded by $\alpha$, we have

$$\begin{aligned}
\sum_{e \in T_L \setminus Q^*} y_e &= \sum_{e \in T_L \setminus Q^*} \sum_{\substack{i: f_i \in Q^*, \\ e \in X(f_i)}} \max\left\{t(f_i) - y_e^i, 0\right\} \\
&\leq \sum_{i: f_i \in R \cap Q^*} \sum_{e \in X(f_i)} \max\left\{t(f_i) - y_e^i, 0\right\} \\
&\leq \sum_{i: f_i \in R \cap Q^*} \alpha = \alpha \cdot |R \cap Q^*|.
\end{aligned}$$

□

Similarly, we can bound the sum over $1 - t(f_i)$ of all edges $f_i \in R \setminus Q^*$. We will see in the proof of the competitive ratio that $1 - t(f_i)$ is the probability for an edge $f_i \in R \setminus Q^*$ to be queried in Random.

Lemma 1.19 *Given an instance of* MST under uncertainty *together with a realization of edge weights, thresholds $t(f_i)$ determined in* Core of Random*, and any feasible query set $Q^*$, it holds that*

$$\sum_{i: f_i \in R \setminus Q^*} (1 - t(f_i)) \leq \frac{1}{2\alpha} \cdot |T_L \cap Q^*|.$$

*Proof.* For an edge $f_i \in R \setminus Q^*$ with $t(f_i) < 1$ we distribute exactly potential $\alpha$ among its neighbors $X(f_i)$ in Algorithm Lines 1 and 2 of the algorithm Core of Random. By Corollary 1.15, $X(f_i)$ is part of the optimal query set $Q^*$. We consider the share of the total potential increase each neighbor receives and distribute the term $1 - t(f_i)$ according

to these shares. Hence,

$$\sum_{i:f_i\in R\setminus Q^*}(1-t(f_i)) = \sum_{i:f_i\in R\setminus Q^*}\frac{1-t(f_i)}{\alpha}\sum_{e\in X(f_i)}\max\{t(f_i)-y_e^i,0\}$$
$$= \sum_{e\in T_L\cap Q^*}\sum_{\substack{i:f_i\in R\setminus Q^*,\\ e\in X(f_i)}}\frac{1-t(f_i)}{\alpha}(y_e^{i+1}-y_e^i)\,. \tag{1.1}$$

In the last equation we have used $y_e^{i+1} = \max\{t(f_i), y_e^i\}$. We consider the inner sum in Equation (1.1) and bound the summand from above by an integral from $y_e^i$ to $y_e^{i+1}$ of the function $\frac{1-z}{\alpha}$. This yields a valid upper bound, as the function is decreasing in $z$ and $t(f_i) = y_e^{i+1}$, unless $y_e^{i+1} - y_e^i = 0$. Hence,

$$\sum_{\substack{i:f_i\in R\setminus Q^*,\\ e\in X(f_i)}}\frac{1-t(f_i)}{\alpha}(y_e^{i+1}-y_e^i) \le \sum_{\substack{i:f_i\in R\setminus Q^*,\\ e\in X(f_i)}}\int_{y_e^i}^{y_e^{i+1}}\frac{1-z}{\alpha}\mathrm{d}z \le \int_0^1\frac{1-z}{\alpha}\mathrm{d}z = \frac{1}{2\alpha}.$$

Now we use this bound in Equation (1.1) and conclude

$$\sum_{i:f_i\in R\setminus Q^*}(t(1-t(f_i)) \le \frac{1}{2\alpha}\cdot|T_L\cap Q^*|\,. \qquad \square$$

Using these two bounds we can calculate the competitive ratio of the algorithm Random.

**Theorem 1.20** *For $\alpha = 1/\sqrt{2}$,* Random *has competitive ratio $1 + 1/\sqrt{2}$, which is approximately* 1.7071.

*Proof.* We consider a fixed realization and an optimal query set $Q^*$, as before. First, note that by Lemma 1.16 all edges queried in the algorithm Framework are in $Q^*$. Now,observe that the increase of potentials in the algorithm depends on the cycles that are closed and thus on the realization, but not on the queried edges. In particular, the edge potentials are chosen independently of the query bound $b$ in the algorithm. Therefore an edge $e \in T_L\setminus Q^*$ is queried with probability $P(y_e \ge b) = y_e$ and an edge $f_i \in R\setminus Q^*$ is queried with probability $P(t(f_i) < b) = 1 - t(f_i)$. Hence, we can bound the total expected query cost by

$$\mathbb{E}[|Q|] \le |Q^*| + \sum_{e\in T_L\setminus Q^*} y_e + \sum_{i:f_i\in R\setminus Q^*}(1-t(f_i))\,.$$

Applying Lemmas 1.18 and 1.19 to this equation yields total expected query cost

$$\mathbb{E}[|Q|] \le |Q^*| + \alpha\cdot|R\cap Q^*| + \frac{1}{2\alpha}\cdot|T_L\cap Q^*|\,.$$

Choosing $\alpha = 1/\sqrt{2}$ yields the desired competitive ratio $1 + 1/\sqrt{2}$ for Random. $\square$

We consider the introductory example described in Figure 1.1 for realization $\mathcal{R}_2$, to show that this algorithm analysis is tight. Random distributes potential $\alpha$ to edge $g$ and thus queries $g$ first with probability $\alpha$ and $f$ first with probability $1-\alpha$. As the realization has the structure $L_f < w_g < U_g \leq w_f$ we need two queries if we query edge $g$ first and one query otherwise. Thus the expected number of queries is $2\alpha + 1 - \alpha$, which is $1 + \alpha$. The optimal query set has size 1, hence Random has expected competitive ratio $1 + \alpha$ for this instance.

## 1.5 Non-uniform Query Costs

We now turn to the problem *MST under uncertainty* in which each edge $e \in E$ has associated an individual query cost $c_e$. Without loss of generality we assume $c_e > 0$ for all $e \in E$ since querying all other edges does not increase the total query cost. We adapt our algorithm Random (Section 1.4) to handle non-uniform query costs achieving the same competitive ratio $1 + 1/\sqrt{2}$ and then show how to derive a deterministic 2-competitive algorithm from it.

Before showing the main results, we remark that for rational query costs the problem can also be transformed into the OP-OP model [GSS16]. This model allows multiple queries per edge and each query returns an open or trivial subinterval (point). Given an uncertainty graph, we model the non-uniform query cost $c_e \in \mathbb{Q}_{>0}$, $e \in E$, in the OP-OP model as follows: Let $m$ be the smallest constant factor that makes all query costs integral. Then querying an edge $e$ returns the same interval for $m \cdot c_e - 1$ queries and returns the exact edge weight upon the $m \cdot c_e$-th query. Then the 2-competitive algorithm for the OP-OP model [GSS16] has a running time depending on the query cost of our original problem.

Theorem 1.21 *There is a pseudo-polynomial, deterministic, 2-competitive algorithm for* MST under uncertainty *with non-uniform query cost.*

### 1.5.1 Randomization for Non-uniform Query Costs

We generalize the algorithm Core of Random (Section 1.4) to the non-uniform query costs model. The adaptation is similar to one for the weighted online bipartite vertex cover problem in [WW15]. For each edge $f_i \in E \setminus T_L$ with query cost $c_{f_i}$ we now distribute at most $\alpha \cdot c_{f_i}$ new potential to its neighborhood $X(f_i)$. We obtain Algorithm 1.4,

Core of Non-uniform Random, by replacing Algorithm Line 1 of Core of Random (Algorithm 1.3) by:

$$\text{Maximize } t(f_i) \leq 1 \text{ s.t. } \sum_{e \in X(f_i)} c_e \cdot \max\{t(f_i) - y_e, 0\} \leq \alpha \cdot c_{f_i} \text{ holds.}$$

We use Non-uniform Random to denote the algorithm Framework used together with

**Algorithm 1.4:** Core of Non-uniform Random

**Input:** A cycle $C_i$ of the algorithm Framework with its edge $f_i$, neighbor set $X(f_i)$, as well as the edge potentials $y_e$ and the query bound $b$.

**Output:** A feasible query set $Q \subseteq C_i$ and a maximal edge.

1 Maximize the threshold $t(f_i) \leq 1$ s.t. $\sum_{e \in X(f_i)} c_e \cdot \max\{0, t(f_i) - y_e\} \leq \alpha \cdot c_{f_i}$;
2 Increase edge potentials $y_e := \max\{t(f_i), y_e\}$ for all $e \in X(f_i)$;
3 **if** $t(f_i) < b$ **then**
4  Add edge $f_i$ to the query set $Q$ and query it.
5 **else**
6  Add all edges in $X(f_i)$ to the query set $Q$ and query them.
7 Return the query set $Q$;

Core of Non-uniform Random. We apply exactly the same analysis as presented in Section 1.4 to prove the competitive ratio of this algorithm. There are non-uniform cost variants of the two lemmas bounding the potential of the edges in $T_L$ and the query probability of edges in $R$.

Lemma 1.22 *Given an instance of* MST under uncertainty *together with a realization of edge weights, the edge potentials after an execution of* Non-uniform Random*, and any feasible query set $Q^*$, it holds that*

$$\sum_{e \in T_L \setminus Q^*} c_e \cdot y_e \leq \alpha \sum_{i : f_i \in R \cap Q^*} c_{f_i}.$$

Lemma 1.23 *Given an instance of* MST under uncertainty *together with a realization of edge weights, the thresholds $t(f_i)$ determined in* Non-uniform Random*, and any feasible query set $Q^*$, it holds that*

$$\sum_{i : f_i \in R \setminus Q^*} c_{f_i} \cdot (1 - t(f_i)) \leq \frac{1}{2\alpha} \sum_{e \in T_L \cap Q^*} c_e.$$

Using the same line of arguments as in the proof of Theorem 1.20, we can derive the following theorem.

Theorem 1.24 *For the non-uniform query cost setting our algorithm* Non-uniform Random *achieves competitive ratio* $1 + \frac{1}{\sqrt{2}}$.

## 1.5.2 Balancing Algorithm

Our polynomial-time algorithm Balance applies the algorithm Framework together with an adaption of the previously described algorithm Core of Non-uniform Random to the deterministic setting. We call this new core algorithm Core of Balance. Erlebach et al. [EHK+08] prove that no deterministic algorithm can achieve competitive ratio less than 2, even in the uniform cost case. Thus we set the parameter $\alpha$ to 1. The goals for the algorithm design are the same as before. We prefer to query the neighbor set of $f_i$, as these edges may appear in several neighbor sets. However, we cannot query the neighbor set, if the additional cost exceeds $c_{f_i}$ to ensure the competitive ratio.

As before we achieve these goals by maintaining an edge potential $y_e$ for each edge $e \in T_L$. We reinterpret it as representing the share of the query cost of edge $e$ for which we have already accounted. As the optimal solution needs to contain either edge $f_i$ or all edges in $X(f_i)$, its cost increases exactly by the smaller of the two costs. We query edge $f_i$, if its query cost is smaller than the not yet covered cost of the neighbors. This is equivalent to a threshold $t(f_i) < 1$. In this case, edge $f_i$ covers an additional cost share of size $c_{f_i}$ in the neighbor set and we increase the edge potentials accordingly. Otherwise, all neighbors $e \in X(f_i)$ are queried.

Similar to the proof of the competitive ratio of algorithm Random, we divide the algorithm's query set into different parts and bound them separately to prove that algorithm Balance is 2-competitive.

Theorem 1.25 *Algorithm* Balance *has competitive ratio* 2*, which is best-possible.*

*Proof.* For some realization, let $Q^*$ denote an optimal query set. Consider the query set $Q$ computed by Balance and let $R := E \setminus T_L$. Then we can split the query set $Q$ into three parts $Q \cap Q^*, (T_L \cap Q) \setminus Q^*$ and $(R \cap Q) \setminus Q^*$. For all edges $e \in T_L \cap Q$ we

**Algorithm 1.5:** Core of Balance

**Input:** A cycle $C_i$ of the algorithm Framework with its edge $f_i$, neighbor set $X(f_i)$ as well as the edge potentials $y_e$.

**Output:** A feasible query set $Q \subseteq C_i$.

```
Maximize the threshold t(f_i) ≤ 1 s.t. Σ_{e∈X(f_i)} c_e · max{0, t(f_i) − y_e} ≤ c_{f_i};
Increase edge potentials y_e := max{t(f_i), y_e} for all e ∈ X(f_i);
if t(f_i) < 1 then
    Add edge f_i to the query set Q and query it;
else
    Add all edges in X(f_i) to the query set Q and query them;
Return the query set Q;
```

have $y_e = 1$, hence,

$$\begin{aligned}\sum_{e\in Q} c_e &= \sum_{e\in Q\cap Q^*} c_e + \sum_{e\in (T_L\cap Q)\setminus Q^*} c_e + \sum_{i:f_i\in (R\cap Q)\setminus Q^*} c_{f_i}\\ &\leq \sum_{e\in Q^*} c_e + \sum_{e\in T_L\setminus Q^*} c_e\cdot y_e + \sum_{i:f_i\in R\setminus Q^*} c_{f_i}\,.\end{aligned}$$

The first term can be trivially bounded by the cost of $Q^*$. For the edges in $R \setminus Q^*$, we charge their full query cost in terms of potential to the edges in the neighbor set. We denote the edge potential at the start of iteration $i$ by $y_e^i$ and denote the edge potential after the last iteration of the algorithm by $y_e$. By Corollary 1.15 we know that $X(f_i) \subseteq Q^*$ for $f_i \notin Q^*$. Thus we can reformulate

$$\sum_{i:f_i\in R\setminus Q^*} c_{f_i} = \sum_{i:f_i\in R\setminus Q^*}\sum_{e\in X(f_i)} c_e\left(y_e^{i+1} - y_e^i\right) \leq \sum_{e\in T_L\cap Q^*} c_e\cdot y_e \leq \sum_{e\in T_L\cap Q^*} c_e\,.$$

For all edges in $T_L \setminus Q^*$, we apply Lemma 1.22 with $\alpha = 1$. Thus we get in total

$$\begin{aligned}\sum_{e\in Q} c_e &\leq \sum_{e\in Q^*} c_e + \sum_{e\in T_L\setminus Q^*} c_e\cdot y_e + \sum_{i:f_i\in R\setminus Q^*} c_{f_i}\\ &\leq \sum_{e\in Q^*} c_e + \sum_{i:f_i\in R\cap Q^*} c_{f_i} + \sum_{e\in T_L\cap Q^*} c_e\\ &= 2\sum_{e\in Q^*} c_e\,.\end{aligned}$$

This factor of 2 is best possible for deterministic algorithms, even in the special case of uniform query costs (cf. Section 1.1, [EHK+08]). □

Remark 1.26 Balance also yields a new 2-competitive algorithm for uniform query cost. Contrary to the deterministic algorithm U-red in [EHK+08] and the one we describe in Remark 1.17, this algorithm does not query pairs of edges. Instead, it either queries edge $f$ or the complete neighborhood $X(f)$ in a cycle.

## 1.6 Matroid Basis under Uncertainty

We consider a natural generalization of MST under uncertainty. Given a ground set of elements and a family of independent sets $\mathcal{I} \subseteq 2^X$, we call a matroid $M = (X, \mathcal{I})$ with an uncertainty interval $A_x$ for each element $x \in X$ instead of the element's weight an *uncertainty matroid*. We define a query and its cost exactly as for the MST problem and refer to the problem of finding a minimum weight matroid basis in an uncertainty matroid using a minimal number of queries as *matroid basis under uncertainty*.

Erlebach et al. [EHK16] show that the algorithm U-red [EHK+08] can be applied to uncertainty matroids with uniform query cost and yields again a competitive ratio of 2. Similarly, our algorithms Random and Balance can be generalized to matroids with non-uniform cost.

Theorem 1.27 *There are deterministic and randomized algorithms for* matroid basis under uncertainty *with non-uniform query cost with competitive ratio* 2 *and* $1 + 1/\sqrt{2} \approx 1.7071$*, respectively.*

In a matroid with known weights we can find a minimum weight basis using greedy algorithms; we distinguish between *best-in* and *worst-out* greedy algorithms (cf. [KV12]). They are dual in the sense that both solve the problem on a matroid and each takes the role of the other on the corresponding dual matroid.

The worst-out greedy algorithm iteratively deletes elements whose weight is too large. We present a worst-out greedy algorithm, Cycle, for uncertainty matroids by merging ideas from the algorithms Random (Section 1.4) and U-red 2 ( [EHK16]). The best-in greedy algorithm adds elements in increasing order of weights as long as the system stays independent. Our best-in greedy algorithm Cut starts with a basis and iteratively considers the set of elements that could replace one element in the basis. Among this set, it chooses the element of smallest weight for the final basis.

Proposition 1.28 *The algorithms* Cycle *and* Cut *are dual to each other in the sense that they solve the same problem on a matroid and its dual.*

We show both algorithms, Cycle and Cut, have competitive ratio 2 for *matroid basis under uncertainty* in the following. They start out with one basis, and then iterate among the family of basis until the optimal one can be identified. This means, the feasibility structure for sets of smaller cardinality does not affect the behavior of the algorithms. Thus, our results do not hold only for matroids, but also for some generalizations of matroids. Namely, set systems whose family of maximal feasible sets equals the basis family of a matroid. We call such set systems *matroid-like*.

Remark 1.29 Our results generalize to matroid-like set systems. Well-known examples that fall into this category are greedoids with the strong exchange property and $\Delta$-matroids (see [KLS91, CK88]).

### 1.6.1 Algorithm Cycle

The algorithm Cycle is inspired by our algorithm Framework as well as the algorithm for uncertainty matroids in [EHK16]. To design a greedy algorithm, we avoid the preprocessing step of the framework and thus do not rely on Assumption 1.10. This makes the algorithm structurally purer and it yields better run times (cp. Chapter 2, Section 2.3) but it may increase the absolute number of queries.

We start with a matroid basis and greedily delete elements, if they cannot improve the basis weight. Analogous to the MST case we define a lower limit matroid basis $B_L$ as a basis for the realization $w^L$, in which all weights of elements with non-trivial uncertainty interval are close to their lower limit, more precisely $w_x = L_x + \varepsilon$ for infinitesimally small $\varepsilon > 0$.

Given an uncertainty matroid $M = (X, \mathcal{I})$, Cycle starts with a minimal lower limit basis $B_L$. We can view this as a first candidate for a minimum weight basis we want to verify. We consecutively add the other elements $f_1, \dots, f_{m-n+1}$ to it in order of increasing lower limit; in the case of ties we prefer the element with the smaller upper limit. In every iteration we maintain a minimum weight basis verified for the already considered element set $X_i := B_L \cup \{f_1, \dots, f_i\}$ with corresponding family of independent sets $\mathcal{I}_i := \{I \cap X_i | I \in \mathcal{I}\}$, i. e. , the matroid $M_i := (X_i, \mathcal{I}_i)$. For each element we add, we consider the minimal dependent set $C$ that is now contained. We query elements from $C$ until we identify a maximal element in this set, by each time choosing an element with maximal upper limit $f$ from $C$ and an element $g \in C \setminus f$ with overlapping uncertainty interval. Note that here element $f$ is not necessarily the just added element $f_i$ in the first

iteration of the while loop, as our uncertainty matroid may not fulfill the equivalent of Assumption 1.10 for matroids.

**Algorithm 1.6:** CYCLE

**Input:** An uncertainty matroid $M = (X, \mathcal{I})$.

**Output:** A feasible query set $Q$.

```
1  Determine lower limit basis B_L; set the temporary basis Γ to B_L;
2  Index all elements in R := X \ B_L by increasing lower limit f_1, f_2, ..., f_{m-n+1};
3  Initialize Q := ∅;
4  for i = 1 to m − n + 1 do
5      Add element f_i to the temporary basis Γ and let C be the occurring minimal
         dependent set;
6      while C does not contain a maximal element do
7          Choose f ∈ C s.t. U_f = max{U_e | e ∈ C};
8          Choose g ∈ C \ {f} with U_e > L_f;
9          Add elements f and g to the query set Q and query them;
10     Delete the maximal element x from Γ;
11 Return the query set Q;
```

The query set the algorithm computes is feasible, as it verifies any element that is deleted is maximal in a dependent set. It terminates, as in each iteration of the while loop at least one element is queried. When all elements in a set $C$ have been queried, we always find a maximal element.

Observation 1.30 CYCLE *is a worst-out greedy algorithm for* matroid basis under uncertainty.

The structure of CYCLE is very similar to our algorithm FRAMEWORK. In particular, we once again maintain a partial solution, i. e. a minimum weight basis verified for a subset of the elements, and extend it by an additional element in every iteration. Hence, it is not surprising, that we can reprove Lemmas 1.12 and 1.13 for the uncertainty matroid setting.

Lemma 1.31 *Given a feasible query set $Q$ for an uncertainty matroid $M = (X, \mathcal{I})$, then the set $Q|_{X_i} := Q \cap X_i$ is a feasible query set for $M_i = (X_i, \mathcal{I}_i)$.*

Lemma 1.32 *Let $B$ be a verified minimum weight basis of the uncertainty matroid $M_i = (X_i, \mathcal{I}_i)$ and let $C$ be the minimal dependent set contained in $B \cup f_{i+1}$. Furthermore, let $h$ be an element with the largest upper limit in $C$ and $g \in C \setminus h$ be an element with $U_g > L_h$. Then any query set verifying a minimum weight basis for $M_{i+1} = (X_{i+1}, \mathcal{I}_{i+1})$ contains $h$ or $g$.*

*In particular, if $A_g$ is contained in $A_h$, any feasible query set contains element $h$.*

Theorem 1.33 Cycle *is* 2*-competitive for* matroid basis under uncertainty.

*Proof.* The query set $Q$ that Cycle computes is built iteratively. In each step we consider an element pair $f, g$ and query the previously not queried part of it. This means we can partition $Q$ into subsets of size at most two and allocate an algorithm iteration to each of them. By Lemma 1.31 any feasible query set must verify a minimum weight basis in that iteration. Furthermore, Lemma 1.32 yields that any feasible query set contains at least one element from the allocated query subset. Using the fact that any query subset has size at most two, this yields that $Q$ is at most twice as large as any feasible query set. □

## 1.6.2 Algorithm Cut

While previous algorithms (U-red [EHK+08], our algorithms) iteratively try to identify the largest-weight element in a dependent set, we now attempt to detect the minimum-weight element that turns an almost inclusion-wise maximal set into a basis. This yields a best-in greedy algorithm based on 'cuts' that finds a minimum weight basis. Interestingly, we show in Section 4.1 that the same algorithm optimally solves the problem of computing the exact weight of the MST next to identifying the MST. As a key to our result, we algorithmically utilize a generalization of the well-known characterization of MSTs through the *cut property* – in contrast to previous algorithms which relied on the *cycle property* (cf. Random, Balance, Cycle, and U-red [EHK+08]).

In our algorithm Cut we choose a particular upper limit basis $B_U$ to start the algorithm. Let $B_U$ be a minimal basis for the realization $w^U$, in which all weights of elements with non-trivial uncertainty interval are close to their upper limit, more precisely $w_x = U_x - \varepsilon$ for infinitesimally small $\varepsilon > 0$. We can view this as a first candidate for a minimum weight basis we want to verify. Let $\Gamma$ initially be this basis $B_U$. Then we delete the basis elements $g_1, \ldots, g_n$ from $\Gamma$ in order of decreasing upper limit; in the case of ties we prefer the element with the larger lower limit. For each element we

delete, we consider the set $S \subseteq X$ of all elements that would complete a basis. We query elements from $S$ until we identify a minimal element in this set. An element is *minimal*, if it has smallest lower limit in $S$ and either trivial uncertainty interval or its upper limit does not exceed the lower limit of any other element in $S$. We will see that this means the element is in a minimum weight basis for any realization. Among the elements in $S$ we always query an element with smallest lower limit $g$ and an element $f \in S \setminus g$ with overlapping uncertainty interval.

**Algorithm 1.7:** Cut

**Input:** An uncertainty matroid $M = (X, \mathcal{I})$.

**Output:** A feasible query set $Q$.

Determine an upper limit basis $B_U$ and set the temporary basis $\Gamma$ to $B_U$;
Index all elements of $B_U$ by decreasing upper limit $g_1, g_2, \ldots, g_n$;
Initialize $Q := \emptyset$;
**for** $i = 1$ *to* $n$ **do**
    Delete element $g_i$ from $\Gamma$;
    Let $S \subset X$ contain all elements $x$ such that $\Gamma \cup \{x\}$ contains a basis;
    **while** $S$ *does not contain a minimal element* **do**
        Choose $g \in S$ s.t. $L_g = \min\{L_e | e \in S\}$;
        Choose $f \in S \setminus \{g\}$ with $L_f < U_g$;
        Add elements $f$ and $g$ to the query set $Q$ and query them;
    Add a minimal element of $S$ to $\Gamma$;
Return the query set $Q$;

The query set computed by the algorithm is feasible as it verifies any element in $\Gamma$ is minimal in a set $S$ and at least one element from the set $S$ is contained in every basis. It terminates as in each iteration of the while loop an element is added or queried. When all elements in a set $S$ have been queried, we always find a minimal element.

Observation 1.34 Cut *is a best-in greedy algorithm for* matroid basis under uncertainty.

We claim that Cut is dual to our algorithm Cycle in the sense that it behaves exactly as Cycle does on the dual matroid. For a given uncertainty matroid $M$, the dual matroid $M^*$ has the same element set and the set of independent sets contains all sets

whose complement contains a basis. Thus a basis of the dual matroid is exactly the complement of a basis of the original matroid.

We will consider the dual matroid with the *inverted weight function*. With this notion we mean for any element $x \in X$ with weight $w_x$ and uncertainty interval $(L_x, U_x)$ we consider the uncertainty interval $(-U_x, -L_x)$ and weight $-w_x$ for the dual matroid.

We first prove that Cut computes a query set verifying a minimum weight basis of the dual matroid for the inverted weight function.

**Theorem 1.35** Cut *computes a* 2*-competitive query set* $Q$ *verifying a minimum weight matroid basis for the dual matroid* $M^* = (X, \mathcal{I}^*)$ *with inverted weight function.*

*Proof.* Cut starts out with an upper limit basis $X \setminus B_U$ of the matroid $M$. According to the inverted weight function, this is a lower limit basis of $M^*$. In the algorithm we sort the elements of $B_U$ by decreasing upper limit. This is the same order as sorting by increasing lower limit for the inverted weight function. The set $S$ we choose is a minimal dependent set, i. e. a cycle, in the dual matroid $M^*$. Exactly as required in Lemma 1.32, we choose the two elements we query such that one has the largest upper limit, i. e. the smallest lower limit according to the inverted weight function, and the other has an overlapping uncertainty interval. Thus, for any element pair we add to the query set $Q$, the optimal query set contains at least one of the two elements.

Therefore Cut computes a query set $Q$ that verifies a minimum weight basis of $M^*$ and has at most twice the size of any query set verifying such a basis. □

**Theorem 1.36** Cut *is* 2*-competitive for* matroid basis under uncertainty.

*Proof.* We need to prove that the query set $Q$ computed by the algorithm Cut verifies a minimum weight matroid basis and has at most twice the size of any feasible query set fulfilling this property. First, we observe that Cut verifies a minimum weight matroid basis of the dual matroid $M^*$ with inverted weight function. The complement of a basis of the dual matroid is a basis of the original matroid $M$. Hence, the algorithm verifies a basis of the matroid $M$ of maximum weight for the inverted weight function. This however, means it verifies a basis of $M$ of minimum weight according to the original weight function.

The line of arguments above shows that any set verifying a minimum weight matroid basis of $M^*$ for the inverted weight function also verifies a minimum weight matroid basis of $M$ for the original weight function and vice versa. Hence, the family of feasible query sets is the same for both problems. As the computed query set $Q$ is at most twice

the size of a feasible query set for a minimum weight matroid basis of $M^*$, it also has at most twice the size of a feasible query set verifying a minimum weight matroid basis of $M$ with inverted weight function. □

Chapter 2

# Computational Experiments for Minimum Spanning Tree with Explorable Uncertainty

In this chapter, we study the minimum spanning tree (MST) problem with uncertain edge weights in experiments. This problem has received quite some attention from the algorithms theory community. We conduct the first practical experiments for MST under uncertainty, theoretically compare three known algorithms, and compare theoretical with practical behavior of the algorithms. Among others, we observe that the average performance and the absolute number of queries are both far from the theoretical worst-case bounds. Our experiments are based on practical data from an application in telecommunications and uncertainty instances generated from the standard TSPLib graph library.

Remark: The results in this chapter are based on joint work with Jacob Focke and Nicole Megow. They are published at the *Symposium on Experimental Algorithms 2017* [FMM17].

The MST problem, one of the most fundamental and practically relevant combinatorial optimizations problems, has been investigated intensively in the uncertainty exploration model from the theoretical perspective. Erlebach et al. [EHK+08] present a deterministic algorithm that achieves the optimal competitive ratio 2. A simplification of this algorithm that omits a repetitive restart by preserving the competitive ratio is presented in Chapter 1 Section 1.6. Also the existence of a dual algorithm is observed. A randomized algorithm with expected competitive ratio of $1 + 1/\sqrt{2} \approx 1.7071$ is given in Section 1.4 of Chapter 1, whereas the best-known lower bound is 1.5. It uses a preprocessing presented in Section 1.2 of the same chapter. The offline problem of finding the

optimal query set for a given realization of edge weights can be solved in polynomial time [EH14].

Compared to the amount of theoretical research, there is a lack of experiments evaluating these results in practice. We are not aware of any experimental results for *MST under uncertainty*. A study on the knapsack problem by Goerigk et al. [GGI+15] seems to be the only work that contains computational experiments conducted in this field.

In this chapter, we theoretically compare three algorithms for *MST under uncertainty*, make practical experiments, and showcase similarities and differences between theoretical and practical observations. In Section 2.1 we present our implementation of the two deterministic algorithms and the randomized algorithm from the previous chapter and compare them theoretically. We also discuss our implementation of the preprocessing we employ for all three algorithms. We show there are instances on which the deterministic algorithms have opposing behavior, meaning that one algorithm is near-optimal and the other shows its worst-case performance. Similarly, instances exist on which the randomized algorithm has opposing behavior to the deterministic algorithms.

We explain the origin of the data for the experiments and the setup for our computational experiments in Section 2.2. We run the experiments on two different data sets. The first set of data is from a telecommunication service provider. It describes a problem that appears when expanding a cable network to a new roll-out area. First, the facility locations are chosen that need to be connected. The exact connection costs between the facilities are unknown and can only be explored through costly field measurements. We find the best MST under uncertainty for these instances. We complement this practical data by a second data set, which we generate based on graphs available in the well-known graph library TSPLib [Rei91]. For both data sets we consider instances where the realizations are uniformly distributed in the uncertainty interval and those with a binary distribution at the two extreme points of the interval.

In Section 2.3 we describe our experimental results and compare them to the theoretical analysis. Our experiments show that the average competitive ratio is small and the total number of queries as well. Both, the competitive ratio and the variance in the size of an optimal solution are far from the worst-case. While theoretically there are instances on which the two deterministic algorithms show opposing behavior, in our experiments their performance is close to equal for all instances. We show that there are instances on which the two deterministic algorithms perform better than the randomized one. Surprisingly, we observe this behavior for the telecommunication data, while for

the TSPLib data the opposite happens and the randomized algorithm has a significantly smaller competitive ratio. For the extreme distribution, where for each edge the realization is at one of the two endpoints of the interval, the preprocessing completely solves all instances of the telecommunication data. For the TSPLib data there are instances where either distribution allows for better performance of the algorithms.

In total, these experiments show that the implementation hurdle is small and our runtimes are reasonably small, even though we did not optimize on it.

## 2.1 Algorithm Introduction and Theoretical Comparison

In this section we compare known algorithms for *MST under uncertainty*. The first (deterministic) algorithm U-RED was introduced by Erlebach et al. [EHK+08]. It achieves the best-possible competitive ratio of 2. In Section 1.6 we present the two deterministic algorithms CYCLE and CUT with competitive ratio 2 for computing a minimum weight matroid basis. Applying CYCLE to compute an MST can be interpreted as a variant of U-RED without repeated restarts and, thus, we consider here only this simplified variant. The randomized algorithm RANDOM we describe in Section 1.4 has competitive ratio 1.7071. Here the best known lower bound is 1.5. For all three algorithms CYCLE, CUT, and RANDOM we apply the preprocessing PREPROCESSING $(\prec_L, \prec_U)$ presented in Section 1.2.

We briefly describe the three algorithms again below and display the pseudo code of our implementation. Recall that we use the following definitions: An edge is *maximal* in a cycle, if it has the largest upper limit and it either has a trivial uncertainty interval or no other upper limit exceeds its lower limit. Symmetrically, an edge is *minimal* in a cut, if it has the smallest lower limit and it either has a trivial uncertainty interval or no other lower limit is less than its upper limit. For a cycle $C$ where no edge is maximal and edge $f$ has largest upper limit, we call the set of all edges $e \in C$ with $U_e > L_f$ the neighbor set $X(f)$ of edge $f$.

**Preprocessing.** The algorithm PREPROCESSING $(\prec_L, \prec_U)$ takes the lower limit order $\prec_L$ and the upper limit order $\prec_U$ defined in Section 1.2. The algorithm repeatedly computes the lower limit tree $T_L$ and the upper limit tree $T_U$ for the two orderings. For

each pair of trees the non-trivial edges in $T_L \setminus T_U$ are queried. This reduces the amount of uncertainty in the problem instance and ensures a structural property crucial for the randomized algorithm: For any cycle closed in the algorithm, any feasible query set contains either the edge with largest upper limit $e$ or all edges with overlapping interval, i. e. whose uncertainty interval contains the lower limit $L_e$.

**Algorithm 2.1:** PREPROCESSING$(\prec_L, \prec_U)$

**Input:** An uncertainty graph $G = (V, E)$.

**Output:** A query set $Q \subseteq E$ and the two trees $T_L, T_U$.

```
1 Q ← ∅;
2 Determine T_L and T_U according to ≺_L and ≺_U respectively using Prim's
    algorithm [AMO93];
3 while T_L \ T_U contains a non-trivial edge do
4     Query all non-trivial edges in T_L \ T_U, and add them to Q;
5     Update T_L and T_U;
6 Return the query set Q and the two trees T_ℓ, T_u;
```

**Deterministic algorithm CYCLE.** The algorithm CYCLE is a worst-out greedy algorithm that is based on the following MST characterization: The largest-weight edge in a cycle is not in any MST. It starts out with a candidate minimum spanning tree and then iteratively considers the other edges by increasing lower limit; in case of ties the the smaller upper limit is preferred. Each additional edge defines a cycle together with the candidate tree. On this cycle, the two edges with largest upper limit are queried repeatedly, until we either verify the additional edge has largest weight or we find an edge of larger weight on the cycle. In the latter case we improve the tree by exchanging the two edges.

**Deterministic algorithm CUT.** CUT is the dual algorithm to CYCLE, that is defined by matroid duality. It uses that the minimum-weight edge in a cut is in an MST. Like in the previous algorithm, CUT starts with a candidate MST, but iteratively considers the tree edges ordered by decreasing upper limit. Here, we break ties by preferring the larger lower limit. Deleting a tree edge defines a cut. On this cut we repeatedly query the two edges with smallest lower limit, until we either verify that the tree edge has the

**Algorithm 2.2:** CYCLE

**Input:** An uncertainty graph $G$ with $G = (V, E)$.

**Output:** A feasible query set $Q \subseteq E$.

```
Execute the algorithm PREPROCESSING, which returns T_L, T_U, Q;
Let f_1, ..., f_{m-n+1} be the edges from E \ T_L ordered by increasing lower limit;
for i = 1, ..., m - n + 1 do
    Add f_i to T_L and let C be the cycle closed by f_i;
    while C does not contain a maximal edge do
        Choose f ∈ C s.t. U_f = max{U_e | e ∈ C};
        Choose g ∈ C \ {f}, with largest upper limit U_g > L_f;
        Query g and f if they are non-trivial and add them to Q;
    Delete a maximal edge of C from T_L;
Return Q;
```

smallest weight in the cut or find an edge of smaller weight to replace the candidate tree edge.

**Randomized algorithm.** The algorithm RANDOM uses the same structure as CYCLE. Starting out with a candidate MST, it iteratively considers the remaining edges and the cycle they close with the tree. However, the algorithm evaluates on each closed cycle more carefully which edges should be queried. The preprocessing yields, that on each such cycle any feasible query set contains either the edge with the largest upper limit, say $f$, or all cycle edges whose intervals overlap with that of $f$. The algorithm either queries the largest edge or all overlapping edges at once. To balance this decision over several cycles closed during the algorithm, RANDOM introduces a potential $y_e$ for each edge. In each cycle additional potential is distributed to all overlapping edges such that they reach an equal level. Depending on the resulting amount of potential, either these edges or the edge with largest upper limit are queried. This decision is taken randomized by comparing the potential to a randomly chosen uniform threshold $b$.

### 2.1.1 Comparing the Deterministic Algorithms

We show that there are instances on which CYCLE and CUT have an opposing performance, meaning that one algorithm is near-optimal and the other shows its worst-case

**Algorithm 2.3:** CUT

**Input:** An uncertainty graph $G = (V, E)$.

**Output:** A feasible query set $Q \subseteq E$.

```
1  Execute the algorithm PREPROCESSING, which returns T_L, T_U, Q;
2  Let g_1, ..., g_{n-1} be the edges from T_U ordered by decreasing upper limit;
3  for i = 1, ..., n − 1 do
4      Delete g_i from T_U and let S be the cut which is created;
5      while S does not contain an always minimal edge do
6          Choose g ∈ S with smallest lower limit L_g = min{L_e | e ∈ S};
7          Choose f ∈ S \ {g} with smallest lower limit L_f < U_g;
8          Query g and f if they are non-trivial and add them to Q;
9      Add a minimal edge from S to T_U;
10 Return the query set Q;
```

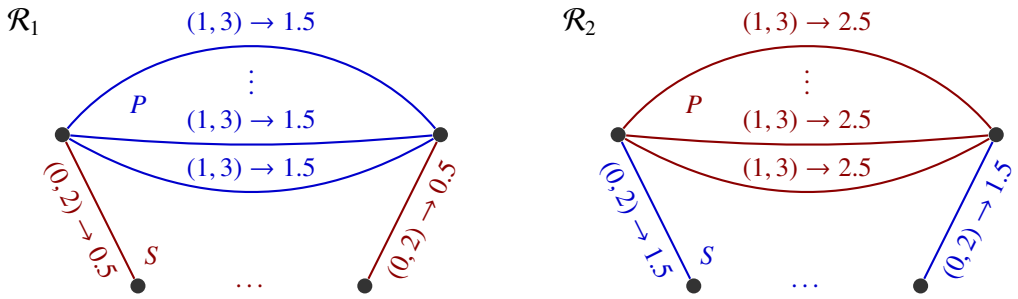



Figure 2.1: Different realizations for the class of uncertainty graphs $SP$ lead to different extremes in the behavior of CYCLE and CUT. Edge labels: $(L_e, U_e) \to w_e$.

performance. Intuitively, the instance is solved by querying the edges of a single cycle $C$ and CYCLE queries pairs of edges on $C$ only. CUT, however, almost exclusively queries pairs with only one edge in $C$. The reverse holds for instances, in which it suffices to query the edges of a single cut.

Our graph class $SP$ consists of a path of edges $S$ and a set of parallel edges $P$, each of which closes a cycle with $S$. We give two realizations $\mathcal{R}_1$ and $\mathcal{R}_2$ in Figure 2.1. For $\mathcal{R}_1$ the set $S$ is the unique optimal query set and a query set is a feasible solution only if it contains $S$. The first cycle closed by the algorithm CYCLE contains $S$ and exactly one edge of $P$. It queries all edges on this cycle, which is a feasible solution of size $|S| + 1$. CUT on the other hand considers cuts of the form $P + \{s\}$ with a non-queried edge $s \in S$.

There are $|S|$ such cuts. For each, Cut queries a pair of edges as long as there are non-queried edges left in $P$. Thus, it queries $|S| + \min\{|S|, |P|\}$ edges. By choosing $S$ and $P$ of appropriate cardinality we can achieve every performance ratio $q \in (1, 2]$ for Cut. In particular, for $|S| \leq |P|$ and $|S| \to \infty$, the performance ratio of Cycle approaches 1 and the ratio of Cut is 2.

The reverse holds for realization $\mathcal{R}_2$. In this case a feasible query set has to contain $P$, Cut finds a solution of size $|P| + 1$, and Cycle queries $|P| + \min\{|S|, |P|\}$ edges.

Observation 2.1 *For any rational $q \in (1, 2]$, there exist a graph in the class S P and a realization such that* Cycle *(*Cut*) is near-optimal whereas* Cut *(*Cycle*) yields a performance ratio of $q$.*

Thus, theoretically the query set sizes can vary greatly for Cycle and Cut. However, we do not observe this behavior for any of the instances in the experiments we present in Section 2.3.

### 2.1.2 Comparing Randomized and Deterministic Algorithms

We show that Random can be optimal for worst-case instances of Cycle and Cut, and – somewhat surprisingly – the reverse is also possible.

Consider a cycle with three edges $f$, $g$, $h$ with uncertainty intervals $(1, 4)$, $(0, 3)$ and $[1, 1]$ respectively. Further, edge $f$ has weight 3 and edge $g$ has weight 1. Then, Random terminates with a single query of either $f$ or $g$, while Cycle and Cut both query $f$ and $g$.

Observation 2.2 *There are instances, for which* Random *finds an optimal solution, while* Cycle *and* Cut *achieve their worst-case ratio of* 2.

A similar instance evokes the reverse performance behavior. Consider a cycle $C$ with $k$ edges $e_i$ with interval $(0, 3)$, one edge $g$ with interval $(0, 4)$ and one edge $f$ with interval $(1, 5)$. We choose the weights as $w_{e_i} = 2$ and $w_g = w_f = 3$. Then Cycle and Cut query only edges $f$ and $g$, which is optimal, but Random yields its worst-case ratio $1 + 1/\sqrt{2}$ for $k \to \infty$.

Observation 2.3 *There exists a family of uncertainty graphs, for which* Cycle *and* Cut *perform optimally, whereas* Random *asymptotically shows its worst case behavior.*

### 2.1.3 Variation in the Size of an Optimal Solution

We investigate the variance of the optimal number of queries, $OPT$, under different realizations for a fixed input instance. We give an example instance in which small perturbations in the realization significantly change the value of $OPT$.

Consider a cycle $C$ of length $m$ consisting of an edge $f$ with uncertainty interval $(1, 4)$ and $m-1$ identical edges $\{g_1, \ldots, g_{m-1}\}$ with uncertainty interval $(0, 3)$ and weight 2. If we set the weight of $f$ to be 3, it suffices to query $f$ and $OPT = 1$. On the other hand, if the weight of $f$ is 2, all edges in $C$ have to be queried and $OPT = m$. Interestingly, we do not observe this large variance in our experiments (see Section 2.3: The optimal solution).

Observation 2.4 *For a fixed uncertainty graph OPT can vary greatly even for minor changes of the underlying realization.*

## 2.2 Experimental Data

First, note that there is an inherent difficulty with practical experiments for exploration uncertainty. For a practical application the uncertainty intervals might be known as well as the exact edge weights of the *queried* edges. To decide the optimal number of queries necessary, one needs the exact edge weights for *all* edges. However, in practice there is no reason to explore additional edges after the solution has been found. Thus, even though we have practical data we need to generate a part of the instance.

**Telecommunication.** For the telecommunication data we are given 5 different graphs of varying size with up to 1000 nodes. For each of them we have two different sets of uncertainty intervals. In the first set, the terrain data, we consider the building cost uncertainty that arises from different terrains. The cost of a connection is limited by the construction cost per meter cable through a field and the construction cost under a paved street times the length of the connection. We draw the exact edge weight *uniformly* distributed in the interval. In this uncertainty setting, exploring the exact weight of an edge represents the time or cost investment it takes to identify the terrain of a particular connection. The second setting assumes that the terrain of the connection is known, but it is uncertain if existing infrastructure is available or not. As a result the interval ranges from almost no building cost due to existing infrastructure to a fixed building

cost, which is roughly known in advance. The exact edge weight follows a *two-point distribution* close to the two endpoints of the interval. We call this the *extreme* setting and maintain the ratio of 20% small weight to 80% large weight that is observed in practice.

**TSPLib.** We consider the 19 graphs for the symmetric traveling salesman problem TSP of the library TSPLib that have at most 100 nodes. They are usually used for TSP computations, but we compute their minimum spanning trees. The library contains the exact edge weights and we need to create corresponding uncertainty intervals. We choose the interval size proportional to the weight of each edge, which is a natural approach that we also observe in the telecommunication data. We experiment with the ratio between interval size and exact edge weight, let us call this ratio d, to generate difficult instances. As before, we consider intervals such that the realization is either uniformly distributed or two-point distributed at the two extremes. For an edge with weight $w$ we draw the lower limit $L$ uniformly at random in $((1-d)\cdot w, w)$ in the uniform case and set the upper limit $U$ to $L + d \cdot w$. In the extremal case we choose the lower limit close to the edge weight $w$ such that $L < w$ or we choose the upper limit $U$ close to $w$ with $w < U$ each with probability $1/2$. Then we choose the other limit accordingly. We computed the average competitive ratio of all three algorithms for the two distributions and various values for $d$ between 0.001 and 0.5 for 190 uncertainty graphs; see Figure 2.9. As we are interested in a worst-case behavior, we choose for our experiments a uniform value $d = 0.065$ for which all algorithms have a rather large competitive ratio.

As one aspect of our experimental analysis, we investigate for a given graph the variance of certain parameters. We distinguish between the two data types: For the telecommunication data the realization inside the uncertainty interval changes, while for the TSPLib data the location of the fixed length uncertainty interval around the also fixed realization changes.

## 2.3 Experimental Algorithm Analysis

For the detailed analysis we draw 100 uncertainty intervals/realizations for each graph in a data set, which yields 4800 instances in total. We perform our experiments with 20 repetitions of Random per instance, as more repetitions did not alter the average per-

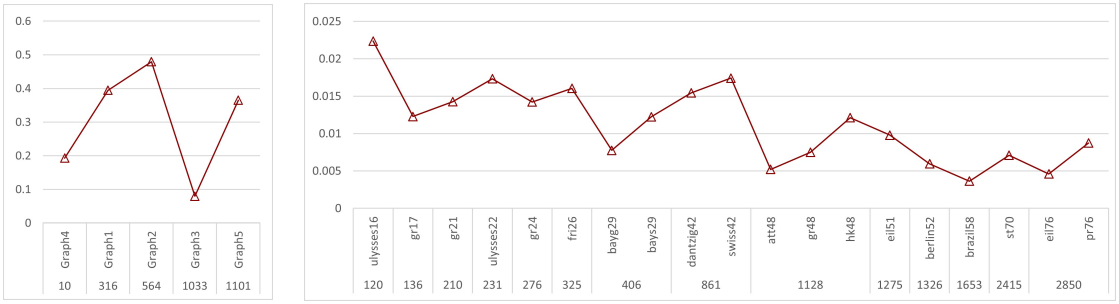



Figure 2.2: Size of the optimal solution $OPT$ divided by the number of edges on the y-axis and the uncertainty graphs sorted by data set and increasing number of edges on the x-axis.

formance. For each of the instances we compute the number of edges, the size of the query set in the preprocessing, the size of the optimal solution, the size of the query set for each of the three algorithms, the runtime of the three algorithms as well as that of the preprocessing. For Random we additionally compute the average number of edges on a cycle closed in the algorithm and the average number of edges on an algorithm cycle that have an uncertainty interval overlapping the one of the edge with largest upper limit. For the latter two parameters, we could not find a relation to the algorithm's performance. We summarize our experimental results in the following subsections. We make our code and the complete input and output data available on [Dat].

### 2.3.1 The Optimal Solution

The size of the optimal solution $OPT$, that is, the minimum number of queries to find an MST, naturally grows with the size of the instance. To analyze a correlation, we consider the number of edges $m$ as the instance size and determine the parameter $OPT/m$; see Figure 2.2. There are instances among the telecommunication data for which the ratio $OPT/m$ is as large as 0.5 and other ones where it is very small. Among this small number of instances the parameter behavior seems arbitrary. For the TSPLib data the ratio $OPT/m$ is a lot smaller and it decreases when $m$ increases. Our theoretical analysis in Section 2.1.3 shows that for a single instance the behavior of this parameter can change between $1/m$ and 1. We do observe great variance for some instances of the telecommunication data, but very small variance for the TSPLib data. The variance is always far from the theoretical maximum variance.

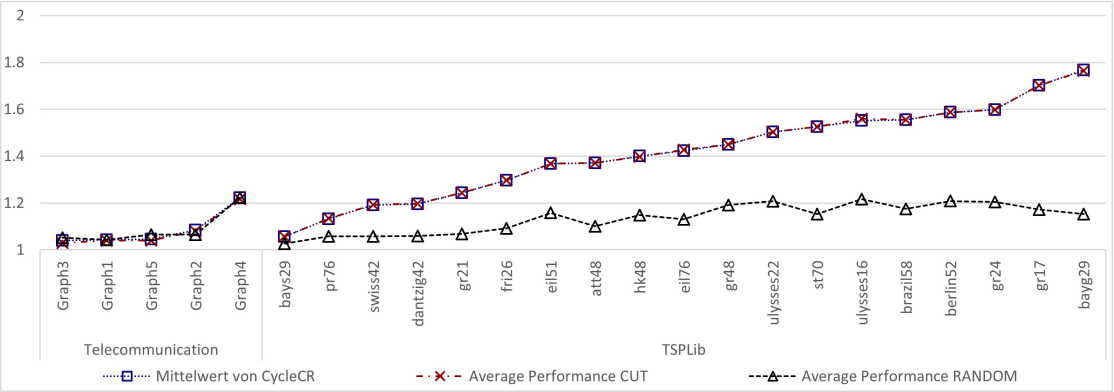



Figure 2.3: Average performance, i. e. the ratio of the algorithm query set size over the optimal query set size, for the three algorithms and the two data sets.

### 2.3.2 Comparing Deterministic and Randomized Algorithms

For the telecommunication data the competitive ratio of all three algorithms has roughly the same average (see Figure 2.3). Averaging over all telecommunication instances Cycle and Cut both have competitive ratio 1.18 and Random has the slightly worse competitive ratio of 1.22. For the TSPLib data, the two deterministic algorithms have equal average competitive ratio 1.37, which is significantly larger than that for the telecommunication data. Random has a notably smaller competitive ratio of 1.11 on average, that is even smaller than the ratio for the telecommunication data.

All average competitive ratios are far from their theoretical worst-case guarantee which is 2 for both deterministic algorithms and approximately 1.7071 for Random. It is somewhat surprising, that despite the significant improvement of our randomized over the deterministic algorithms for the TSPLib data, there is no improvement for the telecommunication data. This means the usefulness of randomization depends on the considered data set. For the telecommunication data our way of randomization may even worsen the performance. This might seem counter-intuitive, but we give a theoretical explanation in Section 2.1.2.

### 2.3.3 Comparing the Deterministic Algorithms

In Section 2.1.1 we show that there can be a large difference between the performance ratio of Cycle and Cut, even to the extreme case where one has ratio 1 and the other has ratio 2. However, as displayed in Figure 2.3, their average performance is identical

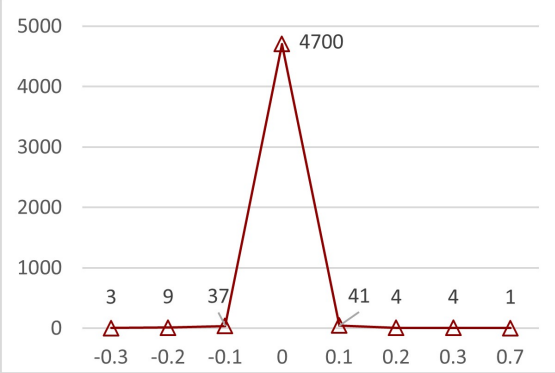



Figure 2.4: Number of instances with performance difference between Cycle and Cut rounded to 1/10.

for all graphs and both data sets. On an instance by instance comparison, the two ratios are equal for 98% of all instances we evaluate (cp. Figure 2.4). The largest difference between performance ratios we observe in our experiments are seven instances with difference 0.33 and one with difference 0.7.

### 2.3.4 Variance in Performance

We compare the average performance of an algorithm to the worst performance among the best 25% of performances as well as the worst performance among the best 75% of all performances. Figures 2.5 and 2.6 show that the variance increases with the average performance ratio of an uncertainty graph and it is greater for the deterministic algorithms than for Random. As the variance is equal for Cycle and Cut, we only display the graph for Cycle. For almost every graph individually, the variance between the different instances is very small.

### 2.3.5 Worst-Case Instances

Both deterministic algorithms have competitive ratio 2. In our experiments, this worst-case ratio is attained for some instances for which the optimal query number is at most 10. As displayed in Figure 2.7, for the telecommunication data the worst-case is attained only on the pathological example of Graph 4 consisting of a single cycle. However, the 7 smallest of the 19 graphs of the TSPLib data showcase instances with performance ratio 2. There are graphs, for which more than half of the instances showcase a worst-case ratio 2, but for others it is only a small percentage. The number of

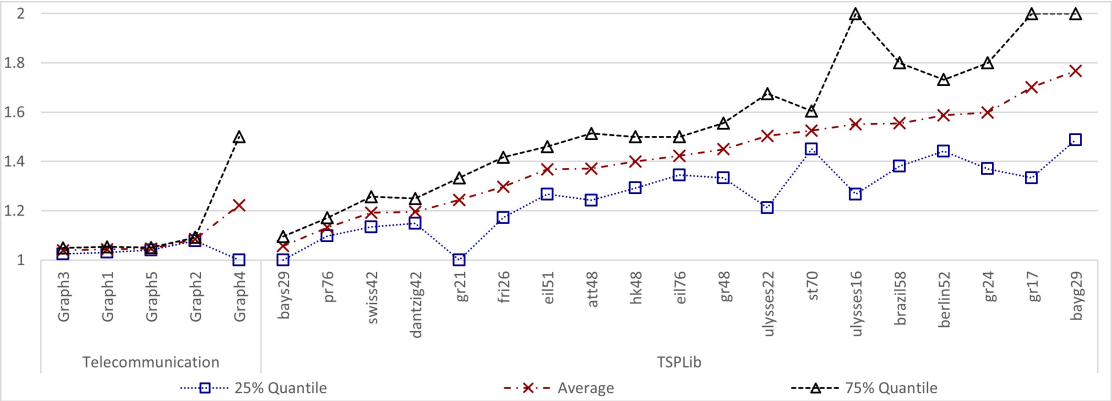



Figure 2.5: Variance of the average performance of CYCLE for each uncertainty graph by displaying the 25% quantile, the average, and the 75% quantile of the algorithm performance.

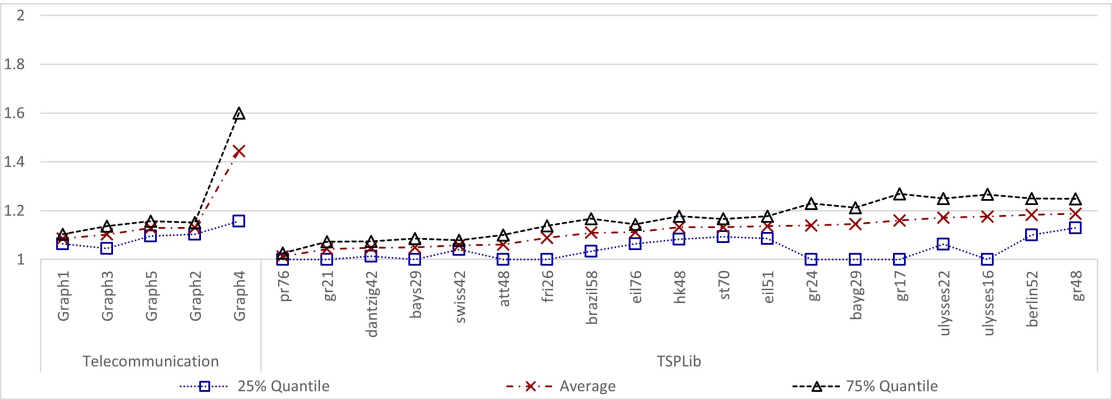
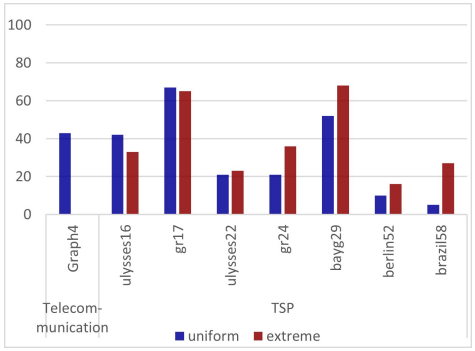



Figure 2.6: Variance of the average performance of RANDOM for each uncertainty graph by displaying the 25% quantile, the average, and the 75% quantile of the algorithm performance.

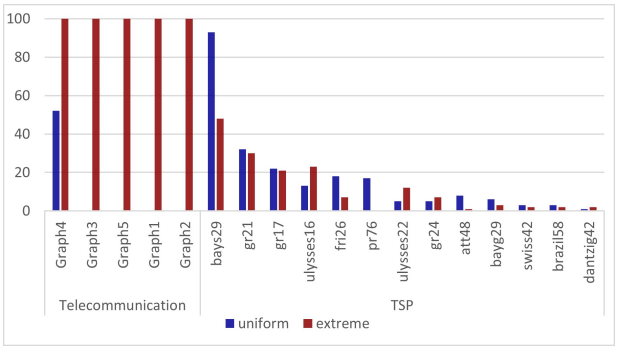
cases of ratio 2 roughly decreases with the number of edges in the graph. This is not symmetric to the case of performance ratio 1. There are more instances and more graphs for which there are instances which the algorithms solve optimally.

### 2.3.6 Comparing the Distributions

PREPROCESSING $(\prec_L, \prec_U)$ solves all telecommunication data instances with extreme distribution. We prove this theoretically in Section 1.2 and observe that it is due to the interval structure and not the distribution. In general, the share of instances solved by the preprocessing significantly increases from uniform to extreme distribution. For the




(a) Number of instances with competitive ratio 2.




(b) Number of instances with competitive ratio 1

Figure 2.7: Instances with best-possible and worst-possible competitive ratio.

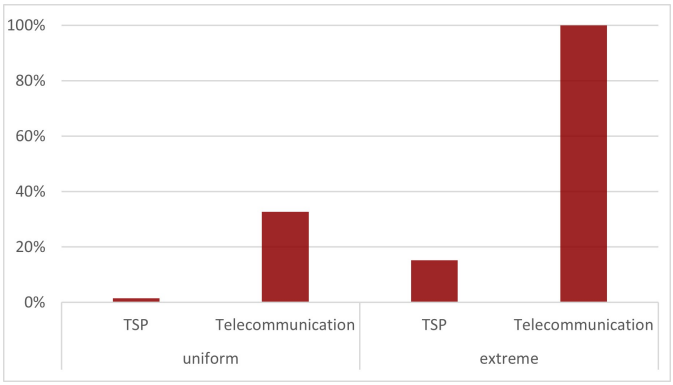



Figure 2.8: Share of instances solved by the preprocessing

TSPLib data the share increases from 0.014 to 0.15 and for the telecommunication data it is 0.33 for the uniform and 1 for the extreme distribution, as displayed in Figure 2.8.

Additionally, we observe that the absolute number of queries almost always decreases, when changing from uniform to extreme distribution for all telecommunication instances and all algorithms. However, for the TSPLib data the behavior varies and for each graph there are instances where the uniform distribution has a smaller query number and others where the extreme distribution has a smaller query number.

### 2.3.7 Interval Size

To create the TSPLib data, we experimented with different interval sizes. Figure 2.9 shows that the algorithms' performance changes greatly with the chosen ratio $d$ of interval size over edge weight. For very large parameter $d$, almost all intervals overlap

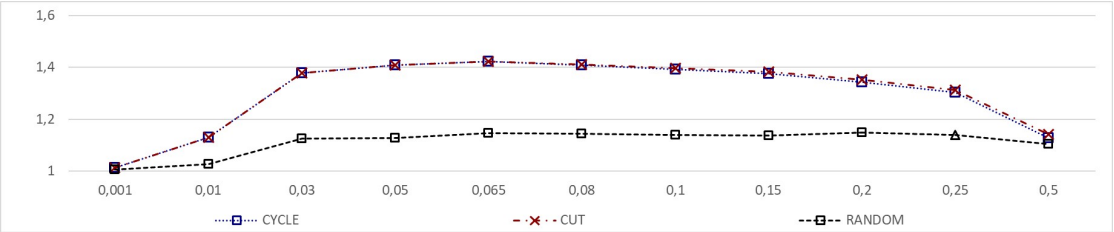



Figure 2.9: Average performance of the three algorithms for the TSPLib data for different values of the parameter $d$ = *interval size over exact edge weight*.

and their edges must be queried. For very small $d$, however, only few intervals overlap and almost no queries are required. This is true for any algorithm, and thus, it explains why Cycle, Cut and Random have an average competitive ratio close to 1 for very small and very large $d$.

### 2.3.8 Runtime

We run our experiments on a Linux system with an AMD Phenom II X6 1090T (3.2 GHz) processor and 8 GB RAM. Together, the three algorithms take about 1200 milliseconds to compute. The preprocessing dominates the runtime with a duration of 770 milliseconds on average. On a one-by-one comparison Cycle and Random have similar average runtimes of around 20 milliseconds, but Cut's average runtime is around 350 milliseconds. As expected, the runtime increases with the graph size. In total, our data set of 400 instances up to a size of 70 vertices or 3000 edges can be generated and solved in roughly four hours. As we did not optimize the implementation in terms of runtime, we expect that also larger instances can be solved in reasonable time.

**Algorithm 2.4:** RANDOM

**Input:** An uncertainty graph $G = (V, E)$.

**Output:** A feasible query set $Q$.

Execute the algorithm PREPROCESSING, which returns $T_L$, $T_U$, $Q$;
Set the temporary graph $\Gamma$ to $T_L$ and index the edges in $R := E \setminus T_L$ by increasing lower limit $f_1, \ldots, f_{m-n+1}$;
Initialize $y_e = 0$, $\forall e \in E$, and choose $b$ uniformly at random in $[0, 1]$;
**for** $i = 1$ *to* $m - n + 1$ **do**
    Add edge $f_i$ to the temporary graph $\Gamma$ and let $C_i$ be the unique cycle closed;
    Let the neighbor set $X(f_i)$ be the set of edges $g \in T_L \cap C_i$ with $U_g > L_{f_i}$;
    **if** $X(f_i)$ *is not empty* **then**
        Maximize the threshold $t(f_i) \leq 1$
        s.t. $\sum_{e \in X(f_i)} \max\{0, t(f_i) - y_e\} \leq 1 + 1/\sqrt{2}$;
        Increase edge potentials $y_e := \max\{t(f_i), y_e\}$ for all edges $e \in X(f_i)$;
        **if** $t(f_i) < b$ **then**
            Add edge $f_i$ to the query set $Q$ and query it;
        **else**
            Add all edges in $X(f_i)$ to the query set $Q$ and query them.
    **while** *no edge in the cycle* $C_i$ *is known to be maximal* **do**
        Query an edge $e \in C_i \setminus Q$ with largest upper limit $U_e$ in $C_i$ and add it to the query set $Q$;
    Delete a maximal edge from $\Gamma$;
Return the query set $Q$;

Chapter 3

# Limits of Optimization with Explorable Uncertainty

In this chapter we consider set systems with uncertain element weights. In general, there is no constant-competitive algorithm [EHK16], so we investigate arbitrary, but fixed set systems. We provide a full combinatorial characterization of set systems that allow competitive ratio 1 and 2. In particular, we prove set systems allow a 2-competitive algorithm if and only if they are matroid-like, i. e. their family of maximal sets equals the basis set of a matroid. For any other set system there cannot be an algorithm with competitive ratio $c < 3$.
We also present non-constant lower bound constructions for two special cases of set systems: matching with uncertain edge weights and knapsack with uncertain element profits. For linear programming we consider uncertain coefficients of the objective function and describe a geometric lower bound construction.

Remark: The results in this chapter are based on joint work with Nicole Megow and Martin Skutella.

Optimization with explorable uncertainty expands far beyond minimum spanning trees and matroids under uncertainty. In this chapter we focus on the limits of optimization under explorable uncertainty, proving strong lower bounds on the competitive ratio for various problem classes. This is in sharp contrast to the previous chapters, where we described algorithms with a small, constant competitive ratio.

We consider set systems with explorable uncertainty as a generalization of matroids. This also generalizes many other well-known problems such as knapsack, matching, and shortest paths. A set system consists of a ground set of elements of uncertain weight together with a family of sets. We aim to find a maximum weight set of this system. In [EHK16] it is shown, that no constant-competitive algorithm exists for this general problem class. Here, the adversary is very powerful, as it chooses the set system, the

uncertainty intervals and the realization. We consider arbitrary, but fixed set systems and thus restrict the adversary to choosing the uncertainty intervals and realization. This approach has also been considered in algorithmic game theory. For congestion games, it has been shown that convergence of best-response strategies and the occurrence of the Braess paradox depends on the combinatorial structure of the strategy spaces [ARV08, FGH+15].

We call a set system *matroid-like*, if its family of inclusion-wise maximal sets equals the basis set of a matroid and show competitive ratio 2 can be attained only for matroid-like set systems. We study parametrized algorithms and lower bounds using $d$, the largest cardinality of a set, as a parameter. Erlebach et al. [EHK16] show a lower bound of $2d$ on the competitive ratio and an algorithm matching this bound. A well-known family of set systems are matchings with uncertain edge weights. Using the parameter $d$ to describes the largest cardinality of a matching, we establish a tight lower bound of $2d$ on the competitive ratio, even for instances of bipartite matching.

A second family of set systems is described by the knapsack problem with uncertain profit of the elements. It has previously only been studied in the uncertainty exploration model where a fixed budget for the query cost is given and the weights are uncertain [GGI+15]. We require to find the optimal knapsack packing and minimize the query cost, as before. For any two primes $r_1 \neq r_2$ we give a problem-specific lower bound instance of knapsack with uncertain profits for which no algorithm uses less than $(r_1 + r_2) \cdot OPT$ queries.

Linear programs with uncertain objective function have not been studied to the best of our knowledge. Given a constraint system and an objective function with uncertainty intervals for its coefficients, we aim to find an optimal solution of this linear program querying as few coefficients of the objective function as possible for their exact value. This generalizes set systems, which means for arbitrary dimension $d$ there is no algorithm with competitive ratio smaller than $d$, which is trivially tight. We give a new, geometric proof of this lower bound. From the geometric interpretation we develop a relaxed model, where queries for arbitrary convex combinations of the coefficients are allowed. However, we show even in this new model no algorithm has competitive ratio less than $d$, the number of coefficients in the objective function.

This chapter is organized as follows: In Section 3.1 we show the combinatorial structure of set systems yields lower bounds for the competitive ratio. We also give parametrized lower bounds for matching and knapsack as special families of set sys-

tems. In Section 3.2 we consider the even more general structure of linear programs with uncertain objective function.

## 3.1 Set Systems

We consider set systems with explorable uncertainty as a generalization of matroids. A set system $(E, \mathcal{F})$, consists of a ground set of elements $E$ together with a family $\mathcal{F} \subset 2^E$ of sets and a positive weight function $w : E \to \mathbb{R}_+$. We aim to find a maximum weight set of this system. In the uncertainty setting, we replace the weight of each element by an uncertainty interval $A_e$. Let $\mathcal{B}$ be the set of all sets of maximum cardinality, we call these sets *basis*. The basis set together with all other inclusion-wise maximal sets forms the set of maximal sets $\mathcal{M}$.

Remark 3.1 Alternative to the restriction to positive weight functions, we could also modify the model to allow negative weights. However, in this case we need to restrict to finding a maximum weight maximal set. By inverting all weights, this is the cheapest set problem.

Set systems are a generalization of minimum spanning tree with positive weight function, as we can define the cycle-free edge sets as the set family and invert the weight function such that a spanning tree with largest weight is an MST for the original weight function. Thus, the lower bound construction in [EHK+08] proves that if we allow closed uncertainty intervals, the competitive ratio is unbounded. Consequently, we restrict to open or trivial uncertainty intervals, as in the previous section. We use $A \triangle B$ to denote the symmetric difference between two sets, that is the union of the two sets $A \setminus B$ and $B \setminus A$.

### 3.1.1 Set Systems with Competitive Ratio at most 2

Set systems are a generalization of minimum spanning trees. Thus the lower bound extends and there cannot be a deterministic algorithm with competitive ratio $c < 2$ or a randomized algorithm with competitive ratio $c < 1.5$. We strengthen this statement and show that the same bounds hold for almost all fixed set systems if the adversary can only choose the uncertainty intervals and the realization.

Theorem 3.2 *For any set system $(E, \mathcal{F})$ with more than one maximal set, there is a set of uncertainty intervals and weights, such that no deterministic algorithm has competitive ratio $c$ for $c < 2$ and no randomized algorithm has expected competitive ratio $c < 1.5$.*

*Proof.* Choose two maximal sets $A, B, A \neq B$ which maximize the size of $A \cap B$. Then there is at least one element $a \in A \setminus B$ and one element $b \in B \setminus A$. We choose the uncertainty intervals of all elements, such that only the two sets $A$ and $B$ are candidates to have maximum weight. For this let $\varepsilon > 0$ be so small, that $|A \triangle B| \cdot \varepsilon < 1$ holds and let the weights be

$$w_e = \begin{cases} 0 & \forall e \in E \setminus (A \cup B) \\ 10 & \forall e \in A \cap B \\ \varepsilon & \forall e \in (A \triangle B) \setminus \{a, b\} \end{cases} \qquad A_a = (0, 3) \qquad A_b = (2, 5).$$

We define a constant $x$ to be $10 \cdot |A \cap B| + 2$. Then the set $B$ has weight strictly larger than $x$, independent of the weight of element $b$. We first show that for any realization either the set $A$ or the set $B$ has maximum weight. Any inclusion-wise maximal set $C$, which does not contain the complete intersection $A \cap B$, has weight less than the maximum sum of all elements minus 10. This is $10 \cdot |A \cap B| + 3 + 5 + 1 - 10 = x - 3$, which is strictly smaller than $x$. Hence, it has smaller weight than the set $B$ independent of the realization. Thus, the set $C$ is not a candidate to have maximum weight. Observe furthermore, that as $A$ and $B$ have maximal intersection by assumption, any set containing the complete intersection additionally contains only elements of weight 0. Thus, these sets also have weight less than $x$ and consequently also less than $B$. Hence, either the set $A$ or the set $B$ has maximum weight for this interval set.

The weight of set $A$ is strictly less than $x + 2$ and the weight of set $B$ is strictly larger than $x$. Thus, if either element $a$ has weight less than 1 or element $b$ has weight larger than 4, one element suffices to prove $B$ is the set of maximum weight in the independence system. We define two realizations, such that for each, querying one element solves the instance but querying the other doesn't:

$$\mathcal{R}_a: \quad w_a = 1 \qquad w_b = 3 \qquad\qquad \mathcal{R}_b: \quad w_a = 2 \qquad w_b = 4.$$

Any deterministic algorithm queries the two elements with uncertain weight either in the order $a, b$ or in the opposite order. Each of the two algorithms, is no better than 2-competitive for one of the two realizations above. Any randomized algorithm queries

element $a$ first with some probability $p \in [0,1]$. It is $p + 2(1-p)$-competitive for realization $\mathcal{R}_a$ and $2p + (1-p)$-competitive for realization $\mathcal{R}_b$. The value $p = 0.5$ minimizes the maximum of these two ratios. For $p = 0.5$, the algorithm is 1.5-competitive for both realizations. Thus, no algorithm has better performance than 1.5. This completes the proof. □

Corollary 3.3 *For minimum spanning tree under uncertainty, this means for any graph with at least one cycle, i.e. that is not a tree, there is a set of uncertainty intervals and weights such that no algorithm is better than* 2*-competitive in the deterministic setting and none is better than* 1.5*-competitive in the randomized setting.*

Remark 3.4 For set systems that have only one maximal set, no queries are necessary and thus there is an exact algorithm.

### 3.1.2 Lower Bound 3 for Non-Matroids

For minimum spanning tree and matroids we achieve competitive ratio 2 with algorithms presented in [EHK+08] and [EHK16]. We show, that once the family of maximal sets does not induce a matroid, there is no 2-competitive algorithm. In the classical book by Schrijver [Sch02] we find the following characterization of maximal sets $\mathcal{M}$ of a matroid:

$$\forall A, B \in \mathcal{M}, a \in A \setminus B \text{ exists } e \in B \setminus A \text{ for which } (B \setminus e) \cup a \in \mathcal{M}. \tag{3.1}$$

Note that this means in particular, that all maximal sets have the same size. Our construction uses a pair of maximal sets with maximal intersection that violate this axiom and gives an instance of uncertainty intervals for the elements. We use $k$ to denote the size of the symmetric difference between these two sets.

Theorem 3.5 *Let $(E, \mathcal{F})$ be a set system and $\mathcal{M}$ its set of maximal sets. If Equation (3.1) is violated by a pair of maximal sets $A, B \in \mathcal{M}$ with $|A \triangle B| = k$ and for any maximal set $A \cap B \subseteq C$ holds $|C \cap (A \cup B)| < \min\{|A|, |B|\}$, then there is a set of uncertainty intervals and weights such that no $c$-competitive algorithm for $c < k$ exists.*

*Proof.* Let $A, B$ be a pair of maximal sets for which Equation (3.1) does not hold and let $a$ be the described element in $A \setminus B$. We first show that we can assume without loss of generality, that the only candidate sets to have maximum weight, are the sets $A$ and $B$. We choose a large constant $M$ and add $M/|A \setminus B|$ to all elements in $A \setminus B$ and add

$M/|B \setminus A|$ to all elements in $B \setminus A$. To elements in $A \cap B$ we add the constant $M^2$. For sufficiently large $M$ this ensures any maximum weight set contains $A \cap B$ and that we can neglect all elements not in $A \cup B$. We add the constant $M + |A \cap B| \cdot M^2$ to the sets $A$ and $B$ and thus do not change their relation. All other maximal sets $C$ with $A \cap B \subseteq C$ have size $|C| < \min\{|A|, |B|\}$, as we neglect elements not in $A \cup B$. Thus, their weight increases by at most $|A \cap B| \cdot M^2 + |M|/|B \setminus A| \cdot |C \cap B \setminus A| + |M|/|A \setminus B| \cdot |C \cap A \setminus B|$. This is at most $|A \cap B| \cdot M^2 + M \cdot (\min\{|A|, |B|\} - 1)/\min\{|A|, |B|\}$. Then, for large enough $M$, none of these sets $C$ is a candidate to be maximal. Consequently, either $A$ or $B$ must be maximal. We neglect the elements in $A \cap B$ in the following, as they do not determine which of the two sets has larger weight. We choose the following set of uncertainty intervals $(L_e, U_e)$ for the elements

$$
\begin{aligned}
&w_e \in (|A|/|B|, |A|/|B| + 1) \quad \forall e \in B \\
&w_e \in (0, 1) \quad \forall e \in A \setminus \{a\} \\
&w_a \in (0, 1.5).
\end{aligned}
$$

Then, the weight of the set $B$ is larger than $|A|$ and the weight of the set $A$ is smaller than $|A| + 0.5$. We now define $k = |A \triangle B|$ many realizations which each have a different element that is contained in any feasible query set. Let realization $\mathcal{R}_x$, $x \in A \triangle B$, have the following weights for a small $\varepsilon > 0$:

$$
w_e = \begin{cases} L_e + \varepsilon & \forall e \in B \setminus \{x\} \\ U_e - \varepsilon & \forall e \in A \setminus \{x\} \end{cases} \qquad w_x = \begin{cases} U_e - \varepsilon & x \in B \\ L_e + \varepsilon & x \in A \end{cases}.
$$

Any algorithm queries one element from $A \triangle B$ as the last element. Let $x$ be this element. Then the algorithm needs at least $k$ queries for the realization $\mathcal{R}_x$, as any feasible query set for this realization contains element $x$. The optimal query set is $\{x\}$, and thus the algorithm is not better than $k$-competitive. This shows, that there cannot be a $c$-competitive algorithm for $c < k$. □

We observe, that for any set system that is not matroid-like, there are at least 3 elements in the symmetric difference $A \triangle B$. If all maximal sets have the same cardinality, we even have $|A \triangle B| \geq 4$.

Corollary 3.6 *Any set system that is not matroid-like allows a set of uncertainty intervals such that the competitive ratio is at least* 3.

Corollary 3.7 *Any set system that is not matroid-like and whose set of maximal sets contains only elements of the same cardinality, allows a set of uncertainty intervals such that the competitive ratio is at least* 4.

Corollary 3.8 *In general, there cannot be any constant-competitive algorithm for finding a maximum weight set system under uncertainty.*

This yields strong lower bounds on optimization in this uncertainty model for a large class of problems. In the next subsections we consider two special cases in more detail: the matching problem and the knapsack problem.

### 3.1.3 Matching under Uncertainty

*Matching under uncertainty* is defined analogue to minimum spanning tree under uncertainty. We are given a graph and instead of edge weights we are provided with uncertainty intervals for all edges, in which the exact edge weight lies. We aim to find an algorithm that needs to query as few edges as possible to find the maximum weight matching. This is special family of set systems.

The maximum weight matching problem without the uncertainty addition is solvable in polynomial time on arbitrary graphs. However, we establish a strong lower bound for *matching under uncertainty* using Theorem 3.5 for set systems. The set of all matchings of a graph defines a set system, where the maximal sets are the inclusion-wise maximal matchings. We note that the structure of the maximal sets is determined purely by the graph structure and not by the intervals. We give a bipartite graph proving that the competitive ratio is unbounded for *matching under uncertainty*. This is particularly surprising, as typically bipartite matching is a very easy problem.

Theorem 3.9 *For any constant $c > 0$ there is a bipartite instance of* matching under uncertainty*, for which no algorithm is c-competitive.*

Consider the graph with $m = 2k$ edges displayed in Figure 3.1. The graph has exactly two inclusion-wise maximal matchings $M_a = \{a_1, .., a_k\}$ and $M_b = \{b_1, ..., b_k\}$, each with cardinality $k$. They are disjoint, and thus $|M_a \triangle M_b| = m$. No other matching has cardinality $k$ and thus we can apply Theorem 3.5. This means there is a set of uncertainty intervals, such that any algorithm has competitive ratio at least $m$, ruling out any constant-competitive algorithm.

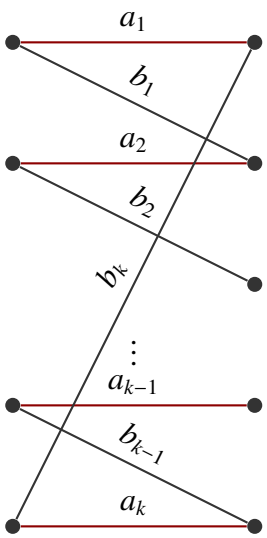



Figure 3.1: Matching graph with competitive ratio $2k$

Remark 3.10 Consider a parametrization by the maximum cardinality $d$ of a matching. Then, the lower bound is $2d$, which is best-possible, as the algorithm for cheapest set under uncertainty by Erlebach et al. [EHK16] is $2d$-competitive. They provide a refined analysis which allows an additive term in the competitive analysis. In this model their algorithm uses no more than $d \cdot OPT + d$ queries. Our lower bound states at least $OPT + 2d - 1$ queries are necessary, which means for this refined analysis the result is not tight.

### 3.1.4 Knapsack with Uncertain Profits

The knapsack problem asks to find a maximum profit packing of items for a fixed size knapsack. Items are defined by a weight and a profit and a feasible packing is one in which the sum of the weights of the chosen items does not exceed the knapsack size. Instead of exact profits, *knapsack with uncertain profits* provides an individual uncertainty interval for the profit of each item. We aim to minimize the number of items we query for their exact profit to determine the most profitable knapsack packing.

We show a non-constant lower bound for *knapsack with uncertain profits*. However, in this case we cannot apply Theorem 3.5 for set systems directly, but we need to design a more sophisticated lower bound instance instead.

Theorem 3.11 *For any two primes $r_1 < r_2$, there is an instance of* knapsack with uncertain profits*, for which no algorithm is $c$-competitive for any $c < r_1 + r_2$.*

*Proof.* Let $r_1 < r_2$ be two primes larger than $c$. We define an instance with two different types of items.

Type $\bar{e}$: $r_2$ items of weight $r_1$ with profit interval $\left(r_1, r_1 + \frac{1}{r_2}\right)$.

Type $\underline{e}$: $r_1$ items of weight $r_2$ with profit interval $\left(r_2 - \frac{1}{r_1}, r_2 + \varepsilon\right)$.

Let the knapsack size $B$ be $r_1 \cdot r_2$. Then all items $\bar{e}$ have joint weight $r_1 \cdot r_2 = B$ and thus form a feasible packing of profit between $r_1 \cdot r_2$ and $r_1 \cdot r_2 + 1$. Another feasible packing are all items $\underline{e}$, as their total weight is $r_1 \cdot r_2 = B$. The cost of this packing is between $r_1 \cdot r_2 - 1$ and $r_1 \cdot r_2 + r_1\varepsilon$.

We choose $\varepsilon$, such that any other feasible packing has profit smaller than $r_1 r_2$. To show this is possible, we first observe, that the ratio $p_i/w_i$ is bounded by $1 + 1/(r_1 r_2)$ for both item types. Furthermore, we observe that any packing that contains some items $\bar{e}$ and some items $\underline{e}$ has total weight at most $B - 1$, as we chose $r_1$ and $r_2$ to be primes. Thus we can bound the total profit of such a mixed packing by

$$\sum p_i \leq \left(1 + \frac{1}{r_1 r_2}\right) \sum w_i \leq \left(1 + \frac{1}{r_1 r_2}\right)(r_1 r_2 - 1) = r_1 r_2 - \frac{1}{r_1 r_2}.$$

Hence, only the two first-mentioned packings are candidates to have maximum profit. For $\varepsilon < 1/(2r_2 r_1)$, we can define $r_1 + r_2$ realizations, that each have a distinct single item necessary and sufficient to prove that the first packing has maximum profit. For realization $\mathcal{R}_x$ let all items $\bar{e}$ have weight close to their lower limit and all items $\underline{e}$ have weight close to their upper limit. However, item $x$ has weight $1 + 1/(r_1 r_2)$, if it is of type $\bar{e}$, and weight $r_2 - 1/(r_1 r_2)$ if it is of type $\underline{e}$. Thus, this yields a lower bound of $r_2 + r_1$ on the competitive ratio of any algorithm. □

Remark 3.12 Consider a parametrization by the parameter $d$, the maximum number of items in a feasible knapsack packing. Then, the cheapest set algorithm by Erlebach et al. [EHK16] is $2d$-competitive. Assuming the twin primes conjecture, which states that there is an infinite number of primes $p$ for which $p + 2$ is also prime [Guy13], our lower bound on the competitive ratio is $2d - 2$, which is almost tight.

Erlebach et al. make a refined analysis of their algorithm by allowing an additive term in the competitive analysis. In this model their algorithm uses no more than $d \cdot OPT + d$ queries. Our lower bound states at least $OPT + r_1 + r_2 - 1$ queries are necessary, which means for this refined analysis a large gap remains.

## 3.2 Linear Programs

In this section we consider linear programs with explorable uncertainty. Given a constraint system $Ax \leq b$ and an objective function $c \in \mathbb{R}$ with uncertain coefficients $c_i \in I_i \subset \mathbb{R}$, each in an individual uncertainty interval. We call $\mathcal{I} = I_1 \times \cdots \times I_d \subset \mathbb{R}^d$ the *uncertainty box*. The aim is to find an optimal solution of this linear program by querying as few coefficients of the objective function as possible for their exact value. As before, we compare any algorithm's performance to the optimal number of queries necessary to determine a vector $x$ satisfying $Ax \leq b$ and maximizing $c^T x$. Contrary to set systems, here, we allow negative weights.

Linear programs have a geometric description as polyhedra. Given a linear program, the inequalitites $A_i x \leq b_i$ describe hyperplanes in $\mathbb{R}^d$. These hyperplanes define a polyhedron $P$, the feasibility region of $x$. We assume $P$ is bounded and thus a polytope. Then $P$ is the convex hull $conv(V)$ of the set of extreme points $V$ of the polytope. For a given objective function $c$, there is always an extreme point $v \in V$ maximizing $c^T x$. We call such an extreme point *maximal* for $c$. For each extreme point $v \in V$, we define a polyhedral cone $C_v = \{y \in \mathbb{R}^d \,|\, y^T v = \max y^T x : x \in P\}$ that contains all vectors $y$ for which this extreme point is maximal. In polytopal geometry, the collection of all these cones $C_v$, $v \in V$, is referred to as the normal fan of the polytope $P$ [Zie12]. We interpret the question of determining the correct extreme point for an uncertain objective function $c$ as identifying a cone $C_v$ in which $c$ lies.

Linear programs are a generalization of set systems, as we can associate to each set its incidence vector. This yields a collection of vectors $V$, whose convex hull defines a polytope $P$. Then, the hyperplanes bounding this polytope form a constraint system $Ax \leq b$. Each element corresponds to one dimension and thus the uncertainty intervals of the elements correspond to the uncertainty intervals of the objective function. The polytopes that are described by independence systems are binary polytopes, as their vertices have only 0 and 1 entries. Linear programs generalize this in several ways. First, they include vectors with integer entries larger than 1. This is a description of independence systems with multiplicities, where each element can occur more than once in a set. Additionally, linear programs also allow real entries.

We observe that, as $P$ is a polytope and thus convex and bounded, no cone can contain an angle of 180 degrees or more.

### 3.2.1 Dimension 1

One-dimensional linear programs describe a very restricted class of linear programs. It contains the class of set systems with a single element, that clearly also have a unique inclusion-wise maximal set. We remark in Section 3.1 that no query is necessary for these set systems if the weights are positive. For linear programs we allow negative coefficients in the objective function and thus this is not always true. Still, there is an exact algorithm. Observe first, that the set of feasible solutions is a segment in the one-dimensional space. Thus it has at most two extreme points, one being maximal for a positive objective function and the other for a negative one.

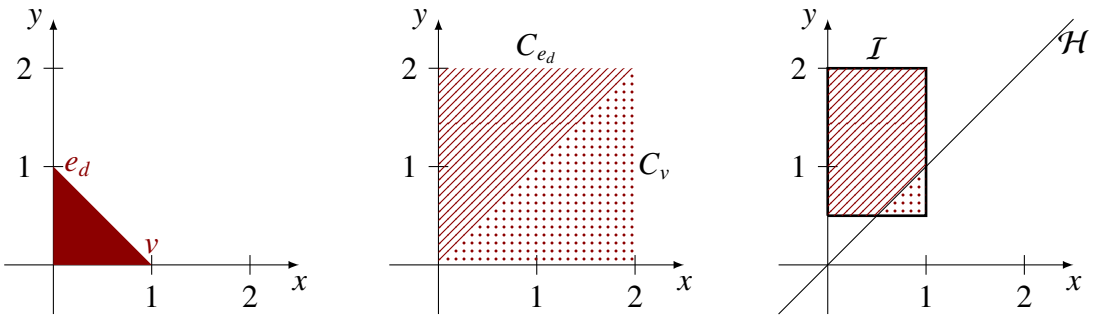
Proposition 3.13 *There is an exact algorithm for finding the optimal vector $x$ maximizing a linear program with uncertain objective function.*

*Proof.* If there is only one feasible point or the uncertainty interval of the objective function is trivial, no queries are necessary to determine the maximal extreme point. The same holds, when the uncertainty interval $I_1$ of the objective function does not contain the origin. Without a query it is clear which is the maximal extreme point. If the origin is contained in $I_1$ and there are two extreme points, then there is a positive and a negative realization for $c$ making opposite extreme points maximal. Hence, any feasible solution needs a query to decide which of the two extreme points is maximal. □

### 3.2.2 Dimension d

For linear programs in arbitrary dimension $d$ there is no general, efficient algorithm. In [EHK16], Erlebach et al. give a lower bound for the cheapest set problem under uncertainty. They allow an additive term in the competitive ratio and show that, if the maximum cardinality of a set in the instance is $k$, there is no algorithm performing better than $k \cdot OPT + k$. Their example for the additive part of the ratio has $2k$ elements and the optimum needs only a single query. Thus, in the context of linear programs this example has dimension $d = 2k$ and yields a purely multiplicative lower bound of $d$ on the competitive ratio.

We give a geometric proof of the same lower bound and depict the main components of the proof for dimension $d = 2$ in Figure 3.2.




(a) 2-Simplex

(b) Conic partition

(c) Uncertainty box $\mathcal{I}$ divided by $\mathcal{H}$ into two parts.

Figure 3.2: Example of Theorem 3.14 for $d = 2$.

**Theorem 3.14** *For any $d \geq 2$ there is a linear program $Ax \leq b$ with $d$ variables defining a polytope $P = conv(V)$ and there is an uncertainty box $\mathcal{I}$ for the objective function $c$, such that any algorithm has competitive ratio at least $d$.*

*Proof.* Let the polytope $P$ be the convex hull of the unit vectors $e_1, \ldots, e_d$, the origin $\mathbb{0}$, and the vector $v = \mathbb{1} - e_d$, where $\mathbb{1}$ is the all-ones vector. Then, for non-negative objective functions either $e_d$ or $v$ attains the maximum. Thus, we consider the hyperplane $\mathcal{H}$ : $x_1 + \cdots + x_{d-1} - x_d = 0$ separating the two cones $C_{e_d}$ and $C_v$ of the conic partition. We design an uncertainty box $\mathcal{I}$ such that this hyperplane cuts off exactly one corner of the uncertainty box and each part of the box is contained in one cone. Let $I_i = (0, 1)$ for $1 \leq i < d$ and $I_d = (d - 1.5, d)$. Then all objective vectors in $\mathcal{I}$ on the same side of $\mathcal{H}$ as $e_d$ are in $C_{e_d}$ and thus are maximized at the extreme point $e_d$. All other vectors lie in $C_v$ and for them $v$ is the optimal extreme point. The objective value of $e_d$ is in the interval $(d - 1.5, d)$ and for $v$ the objective value lies in the interval $(0, d - 1)$.

Now, for a small $\varepsilon > 0$, we consider $d$ realizations that all deviate in one coordinate from the vector $c = (1 - \varepsilon, \ldots, 1 - \varepsilon, d - 1.5 + \varepsilon)$. For realizations $\mathcal{R}_i, 1 \leq i < d$, we set $c_i = \varepsilon$ and for realization $\mathcal{R}_d$ we set $c_d = d - \varepsilon$. Observe that for each of these realizations, querying coordinate $i$ decides that the objective function lies in the cone $C_{e_d}$, but querying any of the other coordinates does not decide it. Any algorithm queries one of these coordinates $i$ last for the realization $c$. Then, the algorithm needs $d$ queries for realization $\mathcal{R}_i$, but the optimal solution uses only one query. □

If we require the origin to be contained in the uncertainty box, we cannot make the construction above, as then the uncertainty box necessarily intersects all cones. However, we can modify the construction to get the same lower bound. We give an intuitive

explanation: Consider a polytope, with a sharp corner (all angles $< 90$) pointing in direction of the all ones vector. Then, the corresponding cone of the conic partition is larger than one hyperquadrant of the space. This means we can design an uncertainty box, for which all its corners apart from one lie in this cone.

Corollary 3.15 *There is a linear program $Ax \leq b$ defining a polytope $P = conv(V)$ and there is an uncertainty box $\mathcal{I}$ for the objective function $c$ with $\mathbb{0} \in \mathcal{I}$, such that any algorithm has competitive ratio at least $d$.*

### 3.2.3 Querying in Arbitrary Directions

Given the strong lower bounds for linear programming with explorable uncertainty, we adapt the model slightly and we add a new type of queries. Querying in arbitrary directions means, we allow queries of the form '$\sum_i a_i x_i = ?$' instead of only axis-parallel queries. These queries can be interpreted geometrically, as cutting the uncertainty box $\mathcal{I}$ with a hyperplane orthogonal to the vector $a$. The result of the query yields the point, where the hyperplane intersects with the direction of the vector. It thus reduces the problem to the intersection of the uncertainty box with this hyperplane. We show, that also in this altered model the lower bound on the competitive ratio is $d$.

Theorem 3.16 *Given a linear program with uncertain objective function, any algorithm determining the maximal extreme point with queries in arbitrary directions has competitive ratio at least $d$.*

*Proof.* Let $P$ be the hypercube polytope $\{x \in \mathbb{R}^d \,\big|\, |x_i| \leq 1\}$ and let $\mathcal{I} = (-1, 1)^d$ be the uncertainty box for the objective function $c$. Consider an arbitrary, fixed algorithm. We design a realization, that it is 0 for the first $d - 2$ queries of the algorithm. For each query we restrict the uncertainty box to the intersection with the hyperplane defined by the query. If answering the $d - 1$-th query with 0 would yield a segment of an axis-parallel ray, we return $\varepsilon$ close to 0 for this query, otherwise the return value is 0. After $d-1$ queries, the uncertainty box is restricted to a segment of a ray. As almost all queries returned 0, the ray passes either through the origin or very close to the origin. Let $r$ be the direction of this ray. The polytope is the hypercube with $\mathbb{0}$ at the center. Thus, its conic partition is separating $\mathbb{R}^d$ into its hyperquadrants. Then, the point $c^-$, where the ray $r$ enters the uncertainty box is in a different cone of the conic partition than the point $c^+$, where the ray leaves the uncertainty box. This means the algorithm cannot decide in which cone the objective function lies without a $d$-th query.

To ensure there is an efficient optimal strategy, we choose $\delta > 0$ so small that the vector $(1 - \delta)c^-$ has length larger than 1. This is possible, as we ensured $c^i$ is not axis-parallel. Let this vector be the realization of the objective function. Then, an optimal strategy is to query in direction $c^-$. This yields exactly the vector length, which is greater than 1 by construction. The hyperplane defined by this query lies outside of the unit ball around the origin. It intersects the uncertainty box $\mathcal{I}$, but none of the coordinate axis are intersected inside the uncertainty box. As the conic partition of the polytope is exactly defined by the hyperquadrants, the hyperplane intersects only one cone $C_v$ inside the uncertainty box. Thus, the extreme point $v$ is proven to maximize the uncertain objective function with a single query. □

Chapter 4

# Interesting Facets of Uncertainty Exploration

In this chapter we study uncertainty exploration beyond the classical model of minimizing the exploration cost to identify a solution that maximizes the uncertain objective function. For MST under uncertainty we show identifying an $\alpha$-approximate MST is as difficult as identifying a true MST, so there is no trade-off. When queries are allowed to return subintervals next to points, surprisingly, randomization does not allow for a competitive ratio smaller than 2. We present a deterministic algorithm for non-uniform query cost achieving the same ratio. Limiting the number of consecutive queries by $r$, and thus enforcing parallel queries, we give a lower bound of $m^{1/r}$ and present a $\max\{2, m/(2r-1)\}$-competitive algorithm.
Furthermore, we present new results for three problems with uncertain feasibility: $k$-th smallest value, sequencing, and knapsack with uncertain weights. We also show an exact algorithm for computing the MST and its weight under uncertainty.

Remark: The results in this chapter are based on joint work with Nicole Megow and Martin Skutella. Section 4.1 and Section 4.2 were published at the *European Symposium on Algorithms 2015* [MMS15] and in *SIAM Journal on Computing* [MMS17]. The latter also contains parts of Section 4.6.

The classical model of uncertainty exploration is minimizing the exploration cost to identify a solution that maximizes the uncertain objective function. In this chapter we study facets of uncertainty exploration that diverge from this classical model. We discuss relaxing the solution quality and limiting the number of consecutive queries that are allowed. Furthermore, we present new results for problems with uncertain feasibility and consider queries that may return subintervals instead of points.

We first consider restrictions of the solutions for *MST under uncertainty*. Khanna and Tan [KT01] study appoximating the sum or average of a set of uncertain elements

and Charikar et al. [CFG+02] present results for AND/OR trees and some generalizations. We show for the model that asks not only to identify the MST, but also to compute its exact weight a 1-competitive algorithm. Then we a relaxed model, slightly deviating from the ones considered in the literature. We call an algorithm $\alpha$-approximate if, for any realization $(w_e)_{e\in E}$, the query set $Q$ found by the algorithm identifies a spanning tree of weight at most $\alpha$ times the weight of an MST. The approximation ratio of an algorithm ALG is the infimum over all $\alpha$, such that ALG is $\alpha$-approximate. Note that an $\alpha$-approximate algorithm does not necessarily compute the approximate weight of an MST. We compare to a weak adversary and consider an optimal solution that has to identify a true, i. e. 1-approximate, MST. We prove, independent of the approximation ratio $\alpha$, no improvement in the competitive ratio is possible in this model. Hence, there is essentially no trade-off between exploration cost and solution quality.

Uncertain feasibility describes the setting where the objective function is known exactly, but input data that decides upon the feasibility of solutions is uncertain. Either, a single feasible solution exists and finding it with little exploration cost is the goal, or there are several feasible solutions. Then, using data exploration we have to find the feasible solution that maximizes or minimizes the optimization goal. We study three problems of this kind: $k$-th smallest value, sequencing, and knapsack with uncertain weights. The first is: Given a set of $n$ elements with uncertain weight, identify an element with $k$-th smallest value using a minimum number of queries that reveal the exact weight of an element. In a non-trivial problem, without queries it is not clear which is an element with $k$-th smallest value and thus a feasible solution. This question was first considered by Kahan [Kah91] for $k = 1$ and $k = \lceil n/2 \rceil$, asking to identify all $k$-th smallest elements. Gupta et al. [GSS16] discuss $k$-th smallest value where one feasible solution has to be identified. They describe an algorithm with additive performance guarantee $OPT + k$ and an example proving no algorithm has multiplicative competitive ratio better than $k \cdot OPT$. This yields lower bound $OPT + k - 1$ for additive performance guarantees. We improve the analysis of their algorithm by one, which shows it is best-possible.

In the sequencing problem, which has previously not been studied according to our knowledge, one aims to sort a set of elements of uncertain weight. We show this is a special case of the minimum spanning tree problem with vertex uncertainty, where each vertex is only known to lie in a given uncertainty area. Erlebach et al. [EHK+08] give a 4-competitive algorithm for this problem. We also observe a relation between

sequencing and vertex cover and employ this to design a 2-competitive algorithm, which is best-possible.

Third, we consider the knapsack problem with uncertain weights. We are given a set of items, each with uncertain weight but certain profit, and a knapsack size. Any set of items whose combined weight does not exceed the knapsack size, is a feasible solution. However, as the weights are uncertain, so is which sets of items are feasible. Contrary to the previous two examples of uncertain feasibility, there is usually more than one feasible item set, so we aim to find the one which maximizes the profit. There seems to be no previous work exactly in this model, but Goerigk et al. [GGI+15] consider a fixed query budget and optimize the quality of a feasible solution that they can guarantee. Also, identifying the optimal solution or an approximate solution can be interpreted as uncertain feasibility. Then, even the feasibility criterion is uncertain without queries. For knapsack with uncertain weights the feasibility criterion is the knapsack size, and this is known. We show a parametrized algorithm for knapsack with uncertain weights. If $d$ is the maximum number of items in a possibly feasible knapsack packing, the algorithm has competitive ratio $d$, which is best-possible.

For both models, uncertain objective function and uncertain feasibility, we consider the aspect of parallelization, as suggested in [EH15]. Given a fixed bound $r$ on the allowed number of rounds of consecutive queries, this forces some queries to be made in parallel. How large is the loss in the competitive ratio which is incurred by this parallelization? A special case is the *offline problem*, where only one round is allowed. Thus, for *MST under uncertainty* we have to choose a query set that suffices to identify an MST independent of the realization. This is related to the verification problem, which asks for a query set that verifies an MST for a given, fixed realization. Erlebach and Hoffman give a polynomial algorithm for the verification problem [EH14] and we show there is also a polynomial time algorithm for the offline problem. For the more general model with $r$ rounds, we prove $m/2$ rounds suffice to achieve competitive ratio 2, where $m$ is the number of edges in the uncertainty graph. A more involved analysis yields an improved algorithm with performance $\max\{2, m/(2r-1)\}$. We also briefly consider the $k$-th smallest value problem and sequencing under the aspect of parallelization.

Finally, we study the OP-OP model proposed by Gupta et al. [GSS16], where a query to an edge $e$ does not necessarily reveal the exact edge weight, but may reveal a new uncertainty interval, a subinterval of the previous one, instead. Here, the input is a sequence of intervals $A_e^1 \supseteq A_e^2 \supseteq \ldots$ and the query set is a multiset of the edges.

Gupta et al. [GSS16] give a deterministic algorithm with competitive ratio 2. We prove randomization does not allow for an improvement over worst-case competitive ratio 2. However, even for non-uniform query cost competitive ratio 2 can be obtained.

This chapter is structured as follows: In Section 4.1 we study *MST under uncertainty* where the exact weight of an MST has to be determined. Then we show identifying an approximate MST is as difficult as a true MST in terms of competitive ratio in Section 4.2. Problems with uncertain feasibility, which includes the $k$-th smallest value problem, sequencing, and knapsack with uncertain weights, are described in Section 4.3. We return to *MST under uncertainty* to study the *offline* problem in Section 4.4. In Section 4.5 we generalize this and consider parallelization, which means querying edges for a fixed number of rounds $r$. Finally, in Section 4.6, we consider the OP-OP model introduced by Gupta et al. [GSS16], where queries can return subintervals.

## 4.1 Computing the Optimal Solution Value

In this section we give an optimal polynomial-time algorithm for computing the exact MST weight in an uncertainty graph. Given an uncertainty graph we aim to identify an MST and compute its weight using a minimum number of queries. We call this optimization problem *MST weight under uncertainty*.

We adapt the algorithm Cut we present in Section 1.6 of Chapter 1. In our algorithm Cut-Weight, Algorithm 4.1, we consider a spanning tree $\Gamma$ and iteratively delete its edges. In each iteration, we consider the cut which is defined by the two halves of the tree and query edges in increasing order of lower limits until we have identified and queried a *minimal* edge in the cut. That means an edge which is in an MST for any feasible realization. Then we exchange the tree edge with the minimal edge.

Theorem 4.1 *The algorithm* Cut-Weight *finds the optimal query set for* MST weight under uncertainty *in polynomial-time.*

*Proof.* We show for every edge we query, that it is in any feasible query set. Assume there is an edge $g$ which contradicts this. Then, let $T$ be the MST which does not contain this edge. We query edge $g$ in the algorithm, when it has the smallest lower limit in a cut $S$. At least one edge $f \in S$ is in the MST $T$ and $T \setminus f \cup g$ is also a spanning tree. As the cut $S$ does not contain a minimal edge when $g$ is chosen in Cut-Weight, edge $f$ has current upper limit $U'_f > L_g$. As we also have $L_g \leq L_f$, this means if the edge weight

**Algorithm 4.1:** CUT-WEIGHT

**Input:** Uncertainty graph $G = (V, E)$.

**Output:** A feasible query set $Q$.

```
1  Find a spanning tree Γ and let Q := ∅;
2  Index the edges of Γ by e_1, e_2, ..., e_{n-1};
3  for i = 1 to n − 1 do
4      Delete e_i from Γ;
5      Let S be the cut containing all edges in G between the two components of Γ;
6      while S does not contain a minimal edge with trivial uncertainty interval do
7          Choose g ∈ S such that L_g = min{L_e | e ∈ S};
8          Query g and add it to Q;
9      Add a minimal edge in S to Γ;
10 Return the query set Q;
```

of $g$ is sufficiently close to its lower limit, we can exchange $g$ with edge $f$ and reduce the weight of the tree $T$. Thus edge $g$ must be in the feasible query set to ensure the spanning tree is minimal.

The query set the algorithm computes is feasible, as it verifies any edge that is chosen for the MST is minimal in a cut. The algorithm queries all edges of the MST, as any edge finally in the tree was a minimal edge with trivial uncertainty interval for some cut in the algorithm. It terminates, because in each iteration of the while loop one edge is queried. At the latest, when all edges in a cut have been queried, we find a minimal edge. It runs in polynomial time, as we query one edge in each iteration and there is a polynomial number of edges. □

It may seem surprising that the cut-based algorithm solves the problem optimally, whereas cycle-based algorithms do not. However, there is an intuitive explanation. The cycle-based algorithms identify the edge of *maximum weight* on a cycle, which is not in the tree. Informally speaking, they have a bias to query edges not in the MST. In contrast, CUT-WEIGHT considers cuts in the graph and identifies the *minimum weight* edge in each cut, which characterizes an MST.

We note that our result immediately extends to matroids.

Theorem 4.2 *There is an algorithm that determines an optimal query set for* matroid basis weight under uncertainty *and computes the exact weight of the basis.*

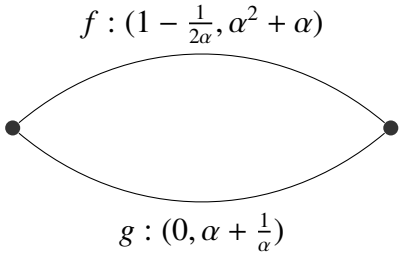



Figure 4.1: Lower bound example for $\alpha$-approximate *MST under uncertainty*.

## 4.2 Approximation

We return to the model, in which we only have to verify an MST, but not compute its weight. We consider an approximate variant in which we relax the requirement that an online algorithm must guarantee an exact MST and allow it to compute an $\alpha$-approximate MST instead. More precisely, an algorithm is $\alpha$-approximate if its query set $Q \subseteq E$ identifies a spanning tree that has weight at most $\alpha$ times the weight of an MST for any realization of edge weights $w_e \in A_e$ for $e \in E \setminus Q$. As before, we evaluate an algorithm's performance by competitive analysis. Note, we only relax the verification requirement for the algorithm, not for the optimum we compare with. The optimal query set still needs to verify an exact MST in the uncertainty graph. We show that despite this significant relaxation of the verification requirements for the algorithm, no performance improvement is possible – for any approximation factor $\alpha$.

Theorem 4.3 *For any $\alpha > 1$, there is no $\alpha$-approximate algorithm for* MST under uncertainty *with competitive ratio $c < 2$. Furthermore, there is no randomized $\alpha$-approximate algorithm for* MST under uncertainty *with competitive ratio $c < 1.5$.*

*Proof.* Consider the uncertainty graph displayed in Figure 4.1 for a fixed approximation ratio $\alpha > 1$. Any deterministic algorithm queries either edge $f$ or edge $g$ first. For each of the two choices, we give a realization which does not give enough information to determine an $\alpha$-approximate MST without a second query, whereas an optimal algorithm can compute the exact MST with a single query. This yields a lower bound of 2 on the competitive ratio.

Realization $\mathcal{R}_1$ has weights $w_f = 1, w_g = 1/(2\alpha)$. Then the optimal query set is $\{g\}$ and has size one. Any algorithm which queries edge $f$ first, cannot verify an $\alpha$-approximate MST after one query. The algorithm has not yet queried edge $g$ but has to

choose an $\alpha$-approximate MST for all possible edge weights of edge $g$. However, edge $f$ is not an $\alpha$-approximate MST for $w_g = 1/(2\alpha)$ and edge $g$ is not an $\alpha$-approximate MST for $w_g = \alpha + 1/(2\alpha)$. Thus the algorithm also needs to query edge $g$ and consequently uses twice as many queries as the optimal algorithm.

Symmetrically we consider the realization $\mathcal{R}_2$, where edge $\{f\}$ is the optimal query set and the edges have weights $w_f = \alpha^2 + 1/\alpha, w_g = \alpha$. Here, an algorithm querying edge $g$ first cannot verify an $\alpha$-approximate MST, as edge $f$ is not an $\alpha$-approximate MST for $w_f = \alpha^2 + 1/\alpha$ and edge $g$ is not an $\alpha$-approximate MST for $w_f = 1 - 1/(3\alpha)$. Hence, again the algorithm needs two queries while an optimal algorithm needs only one query to find an MST.

Any randomized algorithm chooses the algorithm $fg$ with some probability $p$ and $gf$ otherwise. This means it needs $2p + (1 - p)$ queries in expectation for realization $\mathcal{R}_1$ and $p + 2(1 - p)$ queries in expectation for realization $\mathcal{R}_2$. The maximum of these two terms is minimized for $p = 1/2$. Then this algorithm needs 1.5 queries in expectation for each of the two realizations. As before, the optimal query set has size 1 for both realizations, yielding a lower bound 1.5 on the competitive ratio. □

## 4.3 Uncertain Feasibility

In this section we consider three problems where the feasibility of a solution is uncertain. The $k$-th smallest value problem asks to identify the element with $k$-th smallest weight among a set of elements with uncertain weight. Here, there is only one solution that is valid, so there is no cost associated with a solution. We give a tight analysis yielding performance guarantee $OPT + k - 1$ algorithm, where $OPT$ denotes the optimal number of queries. This improves the previous analysis by one. Similarly, we consider the sequencing problem, where we aim to sort a set of elements of uncertain weight. Here we give a best-possible 2-competitive algorithm. Last, we consider the knapsack problem with uncertain weights. We are given a set of items, each with uncertain weight but certain profit, and a knapsack size. Any set of items whose combined weight does not exceed the knapsack size, is a valid solution. However, as the weights are uncertain, so is which sets of items are valid. Here, there is usually more than one valid item set, so we aim to find the one which maximizes the profit. We show a parametrized algorithm for knapsack with uncertain weights. If $d$ is the maximum number of items in a possibly

feasible knapsack packing, the algorithm has competitive ratio $d$, which is best-possible.

### 4.3.1 k-th Smallest Value

We consider the optimization problem *k-th smallest value under uncertainty*. Given a set of $n$ elements, each with an uncertainty interval $A_e$ in which its weight lies. We can query each element for its exact weight and aim to minimize the number of queries until the $k$-th smallest element can be verified. In this whole section we assume $k \leq n/2$, as the cases with larger $k$ can be solved symmetrically.

This problem has previously been studied by Kahan and Gupta et al. Kahan [Kah91] shows that we can find the $k$-th smallest element from a set of $n$ elements with competitive ratio $OPT + 1$, if we ask to output all solutions. Gupta et al. [GSS16] observe that the same holds if we are required to output the lexicographically smallest solution. For the more general case asking for an arbitrary solution, they give an algorithm with performance $OPT + k$, which yields competitive ratio $k$. They also observe that we cannot improve upon competitive ratio $\max\{2, k\}$, if we do not allow an additive term. As their example has $OPT = 1$, this means if we allow an additive term, any algorithm needs at least $OPT + \max\{1, k-1\}$.

We describe the algorithm for the $k$-th smallest value problem presented by Gupta et al. [GSS16]. Our adapted analysis shows the algorithm needs $OPT + k - 1$ queries. This improves the previous result by 1, but is of particular interest, as it yields a tight result.

The algorithm $k$-TH SMALLEST VALUE [GSS16] works in two phases. First, we aim to find the set with the $k$ smallest elements. For this we repeatedly choose the first $k$ elements ordered by lower limits. Among these we then query the element with the largest upper limit. We repeat with choosing the first $k$ elements in order of increasing lower limits, if the set of the $k$ smallest elements is still not defined. Otherwise, we move to the second step, in which we find the largest element among these $k$ elements. Here, we repeatedly sort the elements by increasing upper limit and then query the element with largest upper limit. The algorithm terminates once the $k$-th smallest element has been found and all elements that were queried form the query set $Q$. A formal description of this procedure is given in Algorithm 4.2.

For the algorithm analysis, let $Q^*$ be the optimal query set, $Q = Q_1 \cup Q_2$ be the query sets of the algorithm, and let $f_1, ..., f_n$ be the total order of the elements. The optimal query set $Q^*$ identifies the $k$-th smallest element. Thus it defines the set of the $k-1$ smallest elements and that of the $k$ smallest elements. To make the notation more

**Algorithm 4.2:** $k$-TH SMALLEST VALUE

**Input:** A set of elements $X = \{e_1, ..., e_n\}$ with uncertain weight.

**Output:** A query set $Q$, such that by considering the exact element weight $w_i$ for all elements $e_i \in Q$ and the uncertainty interval for all others defines a $k$-th smallest element.

```
Initialize Q := ∅, Q_1 := Q_2 := ∅;
while we cannot distinguish the k-th smallest elements from the rest do
    Sort the elements by lower limits and reindex such that L_{p_1} ≤ L_{p_2} ≤ ··· ≤ L_{p_n};
    Set S' = {p_1, ..., p_k};
    Order the elements in S' such that U_{q_1} ≤ ··· ≤ U_{q_k};
    Add element q_k to Q_1 and query it;
while we cannot identify the k-th smallest element do
    Query the element with largest upper limit in S' and add it to Q_2;
Set Q = Q_1 ∪ Q_2;
Return the element with largest upper limit in S';
```

compact, denote $\{f_1, \ldots, f_k\}$ by $F_k$.

Lemma 4.4 *For any element $e \in E \setminus F_k$ holds: $e \in Q \Rightarrow e \in Q^*$.*

*Proof.* Let $e$ be an element which is not among the first $k$ elements in the total order and that is queried in the algorithm. Then $e$ is not queried in the second loop of the algorithm, as all these are among the first $k$ elements in the total order. Hence, element $e$ is in the query set $Q_1$ of the algorithm. When $e$ is queried in the algorithm, it has one of the $k$ smallest lower limits. If $e$ is not in the optimal query set $Q^*$, no additional element can be provably smaller. Thus $e$ is among the smallest $k$ elements identified by $Q^*$, which equals the set $F_k$. This contradicts our assumption. □

Lemma 4.5 *For $k \geq 2$ holds: If we have $F_k \subseteq Q$, then $Q^* \cap F_k$ contains at least one of the first $k$ elements.*

*Proof.* We assume for contradiction $Q^* \cap F_k = \emptyset$. Note first, that in [GSS16] they show that Algorithm 4.2 queries at most $OPT + 1$ elements for the second loop of, where we find the element of largest value of a set (cf. their Lemma 6.1). As we assume the optimal query set contains none of the first $k$ elements in the total order, this means the set $Q_2$ of the algorithm has size at most 1. By assumption, all elements in $F_k$ are queried

in the algorithm. Thus, either all $k$ or just $k-1$ elements of the set $F_k$ are queried in the set $Q_1$ and thus in the first algorithm loop.

If $Q_1 \cap F_k$ has size $k$, the uncertainty interval of all these elements overlaps that of $f_k$. Any element with an upper limit smaller than $w_{f_k}$ cannot be in $Q_1$, as it never has the largest upper limit among any set of $k$ elements. To prove $f_k$ has larger weight than all these elements, the optimal query set $Q^*$ needs to contain either $f_k$ or all other elements. As we assume $k \geq 2$, all other elements means at least one element. This contradicts our assumption.

If $Q_1 \cap F_k$ has size $k-1$, the set $Q_2$ must have size exactly 1. When this one element is queried in Algorithm Line 8, we have already identified the $k$ smallest elements, and $k-1$ of them have been queried, as they are in $Q_1 \cap F_k$. The $k$-th smallest element cannot be identified, as otherwise the algorithm would terminate. However, the optimal solution considers the same set of the $k$ smallest elements and does not query any of these elements by assumption. This yields a contradiction. □

Theorem 4.6 *For a set of $n$ elements with uncertain weight, Algorithm 4.2 finds the $k$-th smallest element using at most OPT+ $\max\{1, k-1\}$ queries.*

*Proof.* We split the algorithm query set into the part intersecting $F_k$ and the rest. By Lemma 4.4, the latter part is at most as large as the optimal solution without $F_k$.

$$|Q| = |Q \setminus F_k| + |Q \cap F_k| \leq |Q^* \setminus F_k| + |Q \cap F_k|.$$

For $k = 1$ this completes the proof, as $|Q^* \setminus F_k| \leq |Q^*|$ and $|Q \cap F_k| \leq 1$. Otherwise, Lemma 4.5 yields that $|Q \cap F_k| \leq |Q^* \cap F_k| + k - 1$ holds. Thus, for $k \geq 2$, we have

$$|Q| \leq |Q^* \setminus F_k| + |Q^* \cap F_k| + k - 1 = |Q^*| + k - 1.$$ □

### 4.3.2 Sequencing

We consider the problem of sorting $n$ elements $e_1, ..., e_n$ by increasing weight. Each element has an uncertainty interval $A_i$ and reveals upon a query its exact edge weight $w_i$. We aim to minimize the number of queries until the total order of the elements has been decided and call this problem *sequencing under uncertainty*. First, observe that we can interpret this as a minimum spanning tree problem with vertex uncertainty. Here each element represents a vertex, which lies on the real line in the given uncertainty area. The MST between the vertices, is exactly the tree, where each vertex is connected to

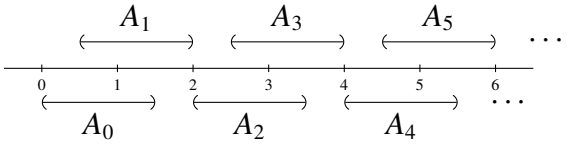



Figure 4.2: Sequencing instance considered in the proof of Theorem 4.8

its nearest neighbors. Knowing the nearest neighbors, in turn defines the sorting of the elements. Erlebach et al. [EHK+08] show a 4-competitive algorithm for finding an MST in the vertex uncertainty setting. This result immediately translates to *sequencing under uncertainty*.

Theorem 4.7 *There is a* 4*-competitive algorithm for* sequencing under uncertainty.

Erlebach et al. [EHK+08] furthermore show for MST with vertex uncertainty, that competitive ratio 4 is best-possible. However, the lower bound construction uses two dimensions, while our sorting construction is on the line. Thus the lower bound does not transfer to *sequencing under uncertainty*. We show a lower bound of 2 on the competitive ratio and argue that this cannot be improved even if we allow an additive term in the performance guarantee. Then, we give an algorithm attaining this bound.

Theorem 4.8 *No deterministic algorithm for* sequencing under uncertainty *has competitive ratio* $c$ *for* $c < 2$*, even if we allow additive terms in the performance guarantee.*

*Proof.* Assume for contradiction, we have an algorithm which queries no more than $c \cdot OPT + d$ elements with $c < 2$ and $d$ arbitrary, where $OPT$ denotes the optimal number of queries. Then, consider an instance with $2\lceil (d+1)/(2-c) \rceil$ elements. Let the uncertainty intervals be $A_i = (i, i+1.5)$ for $i$ even and $A_i = (i-0.5, i+1)$ for $i$ odd, as displayed in Figure 4.2. Then any two consecutive intervals $A_i, A_{i+1}$ overlap for even $i$ and these are the only existing overlaps. Any deterministic algorithm queries for each of these pairs one of the two intervals first. As each two overlapping intervals are symmetric, we can assume without loss of generality that the algorithms always queries the interval with even index first. Now consider the realization, where $w_i = i+1$ for $i$ even and $w_i = i + 0.5$ for $i$ odd. The algorithm queries all elements for this instance, while the optimal query set contains only the elements with odd index, which are $\lceil (d+1)/(2-c) \rceil$ in this case. Thus, the algorithm queries twice as many queries as the optimal solution. This is smaller than $c \cdot OPT + d$ if and only if $OPT$ does not exceed $\lceil d/(2-c) \rceil$. However, by construction this is not the case. Thus, this contradicts the claimed performance of the algorithm. □

We present an algorithm Overlap Vertex Cover, Algorithm 4.3, which we show to achieve best-possible competitive ratio 2. We first create an overlap graph by introducing a node for each element we want to sort and an edge between any two elements with overlapping uncertainty interval. These are the element pairs, for which their order in the sorting is not decided yet. We find a maximal matching of the graph greedily and query these elements. Observe that this subset of the elements is also a vertex cover of the overlap graph. Last we query all elements whose uncertainty interval contains the weight of a different element.

**Algorithm 4.3:** Overlap Vertex Cover

**Input:** A set of elements $X = \{e_1, ..., e_n\}$ with uncertain weight.

**Output:** A feasible query set $Q$.

1 Initialize $Q := \emptyset$;
2 Create an overlap graph $G = (X, E)$ with the elements as vertices and an edge between any two elements with overlapping uncertainty interval;
3 Compute a maximal matching greedily and let $S$ be the set of all vertices in the matching;
4 Add $S$ to $Q$ and query all elements in $S$;
5 **forall** *elements* $e_i \in X \setminus S$ **do**
6  **if** *there is an element* $e_j \in Q$ *with* $w_j \in A_i$ **then**
7  Query element $e_i$ and add it to $Q$;
8 Return the query set $Q$;

Theorem 4.9 Overlap Vertex Cover *computes a feasible query set* $Q$ *for* sequencing under uncertainty *in polynomial time and the size of* $Q$ *is at most twice the size of the smallest feasible query set.*

*Proof.* Observe first, that Overlap Vertex Cover computes a feasible query set. For all elements in $Q$ we reveal their exact element weight and thus their ordering is clear. For any other element $e$ all elements with overlapping uncertainty interval are in $Q$ and their element weight does not lie in the uncertainty interval of element $e$. Thus, we can decide for each of them if its element weight is smaller, equal, or larger than that of element $e$. This means that we can decide upon the position of element $e$ in the ordering.

Consider now a feasible query set $Q^*$ of minimum size. All elements that are added

to $Q$ in Algorithm Line 7 are contained in any feasible query set, as we cannot decide if element $e_j$ occurs before or after element $e_i$ without querying $e_i$. Thus, these element are also in the optimal query set $Q^*$. This means any element in $Q \setminus Q^*$ is queried in Algorithm Line 4. We observe that the set $S = Q \setminus Q^*$ has at most twice as many elements as are edges in a maximal matching of the overlap graph. The number of edges in a maximal matching is a lower bound on the size of a minimum vertex cover, which yields that $S$ is at most twice as large as a minimum vertex cover. Any feasible query set must query at least one element from each overlapping pair. Thus, the minimum size vertex cover of the overlap graph is a lower bound on the size of the query set $Q^*$. As $S$ is at most twice as large as the minimum vertex cover, at least half of its elements are contained in $Q^*$. Consequently we have $|Q \setminus Q^*| \leq |Q^*|$. Thus, we can conclude

$$|Q| = |Q \cap Q^*| + |Q \setminus Q^*| \leq 2|Q^*|.$$

The algorithm runs in quadratic time, as we can compute a maximal matching greedily in linear time and the loop over all elements in $X \setminus S$ takes quadratic time. □

An algorithm for *sequencing under uncertainty* also yields an algorithm for a simple scheduling problem. Given one machine and $n$ jobs with processing times $p_1, \ldots, p_n$ and weights $w_j, \ldots, w_n$, find an ordering of the jobs that minimizes the weighted sum of completion times. In the explorable uncertainty model the coefficients of the objective function are replaced by uncertainty intervals. Thus, we replace the weights by uncertainty intervals $A_1, \ldots, A_n$. Now, the task is to minimize the number of queries for the exact weight of a job to determine an optimal schedule. It is well-known that the optimal strategy schedules the jobs by decreasing ratio $w_j/p_j$, commonly referred to as *Smith's Rule* [Smi56]. Any ordering of the jobs that violates this rule, increases the objective function. Thus, to determine an optimal schedule, any feasible query set must determine the total ordering of the elements according to $w_j/p_j$. Algorithm 4.3 has competitive ratio 2 for sorting elements with explorable uncertainty and thus we have the following theorem for scheduling on one machine.

**Theorem 4.10** *Algorithm 4.3 can be modified to find the optimal schedule for* $1||\sum w_j C_j$ *with uncertain weights with competitive ratio* 2.

**Remark 4.11** For uncertain processing times $p_j$ instead of weights, we get the same result. For a job $j$ with uncertainty interval $(L_j, U_j)$ for its processing time, the ratio $w_j/p_j$ lies in the interval $(w_j/U_j, w_j/L_j)$. If we choose this as the uncertainty interval

for an instance with uncertain weights instead of processing times, and give processing time 1 to the job, this describes the same problem.

### 4.3.3 Knapsack with Uncertain Weights

Similar to the knapsack problem with uncertain profits, which we defined in Section 3.1.4 of Chapter 3, consider a set of $n$ items and a knapsack size $B$. Each item $i$ has uncertain weight in its uncertainty interval $A_i$ and a fixed profit $p_i$. We can query each element for its exact weight. We aim to minimize the number of queries necessary, until the most profitable, valid packing can be identified. For a query set $Q$ a packing is valid, if the sum of its item weights does not exceed the knapsack size $B$ for any realization of item weights $w_i \in A_i$ for $i \notin Q$. We call this problem *knapsack with uncertain weights*.

We first observe that this problem has unbounded competitive ratio. For this, consider all possibly valid knapsack packings sorted by decreasing profit. For the most profitable packing, we need to definitely decide if its weight exceeds the knapsack size or not.

Theorem 4.12 *For any constant $c > 0$ and any algorithm, there is an instance of* knapsack with uncertain weights *for which the algorithm is not $c$-competitive.*

*Proof.* Consider a knapsack problem with $n$ identical items $1, \dots, n$, each with uncertain weight in the interval $A_i = (0, 1)$ and profit $p_i = 1$. For some small $0 < \varepsilon < 1/n$, let the knapsack size be $n - 1 + \varepsilon$. Then the knapsack packing containing all elements is possibly valid. Let the realization be such that there is exactly one item $i$ with weight $\varepsilon$ and all other items have weight $1 - \varepsilon$. Then, packing all elements is valid and the only optimal packing. To verify its validity, it is necessary and sufficient to query item $i$. The algorithm cannot distinguish between the items and thus it queries item $i$ last for some choice of $i$. Thus, it queries $n$ items, while the optimal solution is to query only item $i$. □

As we cannot find a general constant-competitive algorithm, we parametrize our instance. Let $d$ be the size of the largest possibly valid knapsack packing. We first observe, that if there are several disjoint packings that are possibly valid and have the same weight, the competitive ratio is also unbounded. Thus, we further restrict the instance, such that all packings have unique weight. Now, we can apply the cheapest set algorithm and improve its analysis for knapsack problems. By the analysis in [EHK16],

**Algorithm 4.4:** Cheapest Set for Knapsack with Uncertain Weights

**Input:** An set of items with uncertain weight and a knapsack size $B$.

**Output:** A query set $Q$ that verifies a provably maximum profit feasible knapsack packing $P$.

```
1  Let Q = ∅ and i = 1;
2  Sort the possibly feasible packings by decreasing profit P_1, ... P_k;
3  while P_i is not provably feasible and i ≤ k do
4      Query all items in P_i and add them to Q;
5      if P_i is not feasible then
6          Increase i by 1;
7  if i > k then
8      Return there is no feasible knapsack packing;
9  else
10     Return the query set Q and the most profitable feasible packing P_i.
```

the cheapest set algorithm has competitive ratio $d \cdot OPT + d$. We show, for *knapsack with uncertain weights* this competitive ratio is $d$.

Theorem 4.13 *Given an instance of* knapsack with uncertain weights *in which the maximum number of items in a possibly valid knapsack packing is d, then Algorithm 4.4 is d-competitive, which is best-possible.*

*Proof.* Let $Q$ be the query set of Algorithm 4.4 and $Q^*$ the optimal query set. We compare the queries of the algorithm in each iteration to $Q^*$. In each iteration we query a possibly but not provably valid packing $P$. If $P$ is the optimal packing, $Q^*$ contains at least one item of $P$ to prove its validity. If $P$ is not the optimal packing, $Q^*$ contains at least one item to prove $P$ is not a valid packing. The algorithm queries at most $d$ items in each iteration.

By the example in the proof of Theorem 4.12, this algorithm is best possible for the parameter $d$. □

Remark 4.14 Contrary to previous problems, the proof for the algorithm does not rely on the fact that the intervals are open or trivial. Thus, for *knapsack with uncertain weights* we can also allow closed, non-trivial uncertainty intervals.

## 4.4 The Offline Problem

We consider a variant of *MST under uncertainty*, where we cannot await the return value of the queries. Thus, we need to decide for one set of edges to query, such that independent of the realization this edge set will reveal a minimum spanning tree. We call this the *offline* problem, as no decisions can be taken after some edge weights are revealed. It is closely related to the verification problem described in [EHK16], where given an uncertainty graph and a fixed realization, they aim to compute the minimum number of queries necessary to verify a minimum spanning tree. A feasible query set for the offline problem has to verify an MST, independent of the revealed realization. Thus, the union of the solutions of the verification problem over all realizations is a feasible solution for the offline problem. We aim to find the smallest of these solutions. Given an uncertainty graph we would like to find the set of edges we have to query to find an MST independent of the realization. We give an algorithm that finds the edge set we need to query in polynomial time.

We first divide the edges into three sets, those edges that appear in an MST for every realization, those that do not appear in a minimum spanning tree for any realization and those that are sometimes part of an MST. We prove that the latter set is exactly the set of edges that we have to query to solve the problem. We call this algorithm Offline and give a formal description in Algorithm 4.5.

As we do not analyze the competitive ratio of this algorithm, but only it's running time, the restriction to open intervals is not necessary, here. To simplify the notation, we assume all uncertainty intervals are closed. If they were open, we choose a value smaller than the next larger lower limit (larger than the next smaller upper limit) and this way order the weights in the same way they would be ordered by choosing the lower (upper) limit of closed intervals.

We will prove three lemmas about the algorithm to show it finds the optimal query set. We first prove the claim that all edges in $\mathcal{M}$ appear in an MST, independent of the realization. Then we argue that the edges of $\mathcal{N}$ do not appear in an MST of any realization. Third, we show the edges in neither of the two groups are in any feasible query set.

**Lemma 4.15** *Any edge $f \in \mathcal{M}$ is in an MST for any realization.*

*Proof.* Assume there exists a realization for which edge $f$ is not in an MST. Then there exists a cycle, where all edges have edge weight smaller than that of edge $f$. We consider

**Algorithm 4.5:** OFFLINE

**Input:** An uncertainty graph $U = (V, E)$.

**Output:** A provably minimal spanning tree $T \subseteq E$.

```
1 for each f ∈ E do
2     Choose U_f for w_f and L_e for w_e of all other edges;
3     if f ∈ MST then
4         Add f to M;
5     Choose L_f for w_f and U_e for w_e of all other edges;
6     if f ∉ MST then
7         Add f to N;
8 Query all edges in E\(M ∪ N);
9 Run Kruskal's Algorithm to find a minimum spanning tree of G and return it;
```

this cycle and compare the edge weights to the realization chosen in Algorithm Line 2 for edge $f$. In this realization the edge weight of all edges of the cycle apart from $f$ is smaller and that of edge $f$ is larger, thus edge $f$ is also not in an MST for this realization. This is a contradiction to $f \in \mathcal{M}$. □

Lemma 4.16 *Any edge $f \in \mathcal{N}$ is not in an MST for any realization.*

*Proof.* Symmetric to Lemma 4.15, we assume there exists a realization for which edge $f$ is in an MST. Then, edge $f$ is the minimal edge in some cut $S$. When we decrease the weight of edge $f$ to its lower limit and increase the weight of all other edges in the cut $S$, edge $f$ is still minimal in the cut. Thus, it is in an MST, independent of the edge weight of all edges that are not in the cut $S$. This means, $f$ is in an MST for the realization chosen in Algorithm Line 5 for edge $f$. However, this is the only time when $f$ could be added to $\mathcal{N}$, so we have a contradiction to $f \in \mathcal{N}$. □

Lemma 4.17 *For all edges $f \in E\backslash(\mathcal{M} \cup \mathcal{N})$ exists a realization in which $f$ is in any feasible query set.*

*Proof.* We consider the realization described in Algorithm Line 2 and a corresponding MST $T$. Then edge an edge $f \in E\backslash(\mathcal{M} \cup \mathcal{N})$ is maximal in the cycle $C$ in $T \cup f$. If the uncertainty interval of edge $f$ overlaps with any of the edge weights on the cycle, edge $f$ is in any feasible query set. Thus, we assume this is not the case. This means we can reduce the weight of edge $f$ to $L_f$ without changing the MST $T$. If we increase

the weights of all other edges on $C$ to their upper limit, edge $f$ is still maximal on $C$. However, this means $f$ is also maximal on this cycle for the realization considered in Algorithm Line 5, which is a contradiction. □

Theorem 4.18 *Algorithm 4.5 finds the optimal solution for the* offline *variant of* MST under uncertainty *using the minimum number of queries possible. It runs in polynomial time.*

*Proof.* The three lemmas above prove that all edges queried in the algorithm are in any feasible query set and that the queried elements form a feasible query set.

The algorithm iterates over all edges. For each edge, we perform two runs of Kruskal's Algorithm that has running time $O(n \log n)$. This means the for each loop has running time $O(n^2 \log n)$. The last run of Kruskal's Algorithm is dominated by this running time of the for each loop. □

## 4.5 Parallelization

Until now we have analyzed optimization with explorable uncertainty, where we could await the return value of each query before deciding which edge to query next and the one where were we could not await the return value of any query. In this section we show what changes when there is a bound on the number of times we can query edges. This means that sometimes we have to query more than one edge at the same time and that we have to decide which edges to query together. A query set now means to query a specific set of edges of arbitrary size. We can use the weights of the edges in one query set to which edges to choose for the next query set and the total number of query sets is bounded by a parameter $r$. The case we have analyzed until now, is when there are as many rounds allowed as there are elements or edges. Here, we achieve a worst case competitive ratio of 2 for *MST under uncertainty* and algorithm performance $OPT + k - 1$ for *k-th smallest value under uncertainty*.

### 4.5.1 MST with Explorable Uncertainty

For *MST under uncertainty* we have seen two algorithms with competitive ratio 2 and a randomized algorithm with competitive ratio 1.7071 in Chapter 1. In this section, we only consider deterministic algorithms. If there are $m$ edges in a graph, these algorithms

finish after at most $m$ rounds. We first analyze an adaption of the algorithms Framework and Cycle that yields competitive ratio $m/r$. Then, we give a lower bound construction and last we describe an improvement of the previous algorithm.

The algorithm Cycle from Section 1.6 uses at most $m$ rounds to find a minimum spanning tree, as it queries at least one edge in each round. It has competitive ratio 2. We give an adapted algorithm Parallel Cycle that has competitive ratio $\lceil m/r \rceil$ for $r \leq \lceil m/2 \rceil$ rounds and competitive ratio 2 for larger round number $r$. The original algorithm Cycle usually queries two edges per round, but sometimes it only queries a single edge. Our adaption ensures the algorithm queries two edges in every round. Additionally, in the last round we query all unqueried edges. One could consider applying the algorithm Preprocessing from Section 1.2 before executing Cycle. However, in the preprocessing we might query only one edge per round and it is unclear how to adapt the preprocessing. Thus, we do not employ the preprocessing. We use the structure of the algorithm Framework from Section 1.3 which also is the base for the algorithm Cycle. Our adapted algorithm Parallel Cycle starts with a lower limit tree $T_L$ and iteratively adds the other edges by increasing lower limit. In each cycle, we consider the two edges $f$ and $g$ with largest upper limit. If both edges have not been queried, the query set has two edges as desired. However, sometimes the second edge chosen in the algorithm, $g$, has a trivial uncertainty interval. In this case we choose an arbitrary edge with non-trivial uncertainty interval and query it together with edge $f$. We display the pseudo-code in Algorithm 4.6.

**Lemma 4.19** *Given an uncertainty graph $G$ with $m$ edges and a positive integral constant $1 \leq r \leq m$. Algorithm 4.6,* Parallel Cycle*, achieves competitive ratio* $\max\{2, \frac{m}{r}\}$.

*Proof.* We first show that at least one edge from any edge pair $\{f, g\}$ queried in the algorithm is in any feasible query set. Whenever we query an arbitrary edge, this is either already in the temporary graph $\Gamma$, or it has not yet been added. If it is already in $\Gamma$, this does not change the proofs from the algorithm Framework in Section 1.3, as in $\Gamma$ any edge can be queried or unqueried. If we query an edge that is not yet in $\Gamma$, it is added to $\Gamma$ at some point, even though it has already been queried. The order in which edges are added, appears only in the proof of Lemma 1.13. There, we use the fact, that the edges are added according to their original intervals by increasing lower limit. This property has not changed, so the proof goes through as before. In the cycle itself, adding and edge $f$ which has already been queried is the same as if edge $f$ was queried after it was added. Thus, also here the proofs remain valid. This means Parallel Cycle computes

**Algorithm 4.6:** Parallel Cycle

**Input:** An uncertainty graph $G = (V, E)$ and a maximal number of allowed rounds $r$.

**Output:** A feasible query set $Q$.

```
1  Determine lower limit tree T_L and set the temporary edge set Γ to T_L;
2  Index all edges in E \ T_L by increasing lower limit f_1, f_2, ..., f_{m-n+1};
3  Initialize Q := ∅ and k := 0;
4  for i = 1 to m − n + 1 do
5      Add edge f_i to the temporary edge set Γ and let C be the occurring cycle;
6      while C does not contain a maximal edge do
7          if k < r then
8              Choose f ∈ C s.t. U_f = max{U_e | e ∈ C};
9              Choose g ∈ C\{f} with U_e > L_f;
10             if g has trivial uncertainty interval then
11                 Rechoose g as an arbitrary edge with non-trivial uncertainty
                   interval;
12             Add elements f and g to the query set Q, query them and increase k
                by 1;
13         else
14             Add all unqueried elements to the query set Q, query them and
                increase k by 1;
15     Delete the maximal edge in C from Γ;
16 Return the query set Q;
```

a query set $Q$ that verifies a minimum spanning tree and at least one edge from any pair of edges queried at the same time is in any feasible query set.

In each round apart from the last one, exactly two elements are queried that have not been queried previously. In the last round all remaining edges are queried. Thus, if $r \geq \lceil m/2 \rceil$, or if the algorithm terminates before round $r$, in each round two edges are queried. We show the algorithm has competitive ratio 2 in this case, by considering each round individually. If in one round the two edges $f$ and $g$ are chosen as in the algorithm Framework, any feasible query set contains at least one of them by Lemma 1.12 in Section 1.3. Otherwise, the edge $f$ is queried with an arbitrary other edge. In this

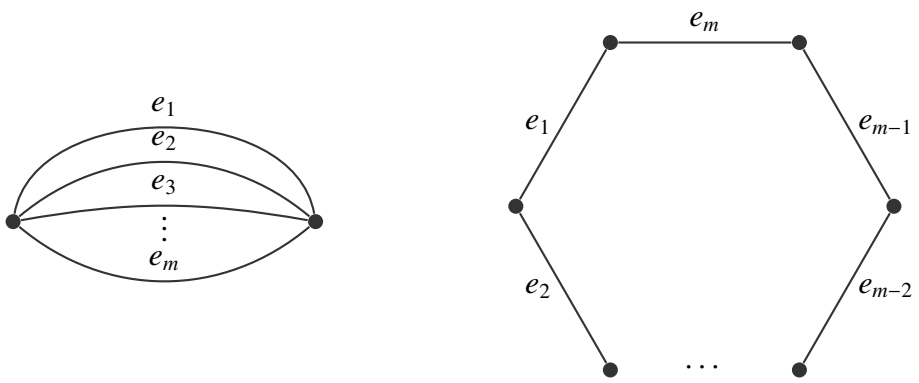



Figure 4.3: Graphs for the lower bound construction.

case, edge $g$ has trivial uncertainty interval. As edge $f$ is not maximal in this cycle, the uncertainty interval of edge $f$ must contain the weight of edge $g$. Then edge $f$ is in any feasible query set by Lemma 1.13. Thus, for any round at least one of the queried edges is in any feasible query set. As no edge with trivial uncertainty interval is queried, the edge sets queried in the algorithm are disjoint. For each round at least one of the queried edges is in any feasible query set, thus the competitive ratio of the algorithm is 2.

If the algorithm proceeds to round $r < \lceil m/2 \rceil$, the optimal query set contains at least $r$ edges. The algorithm may query all edges that have not been queried yet in round $r$ and thus it queries at most $m$ edges in total. This yields competitive ratio of $m/r$ for this case. □

To construct a lower bound, we consider the graphs displayed in Figure 4.3, where edges are labeled by increasing lower limit. We describe a family of algorithms for such graphs and show it contains the best-possible algorithms. We describe the construction for the graph on two nodes, but the same ideas can be applied to the cycle graph using matroid duality as described in Section 1.6. Without loss of generality we assume the smallest upper limit is larger than the lower limit of edge $e_m$. Otherwise, we can simplify the graph and delete all edges with lower limit larger than the smallest upper limit, as they are definitely not in a minimum spanning tree. We call the resulting set of uncertainty intervals mutually overlapping.

Our algorithm family Parallel Algorithm on Two Nodes, Algorithm 4.7, queries the edges ordered by increasing lower limits. Given a fixed number of rounds $r$, it receives an index sequence $0 = \ell_0 \leq \ell_1 \leq \cdots \leq \ell_r = m$ and it queries edges, such that after round $i$ the $\ell_i$ edges with smallest lower limit have been queried. The special graph structure means we must query edges until we find the edge with minimal weight

to identify an MST. Thus, we need to find an edge whose upper limit is at most as large as the smallest lower limit of the other edges in the graph, i. e. a minimal edge. By assumption, the edges are mutually overlapping. Thus, there are two possibilities to show this. Either we query the edge itself and all edges whose interval contains the edge's weight, or we query all edges but the minimal one and their weight lies above the upper limit of the minimal edge. The latter is only possible, if the minimal edge has the unique smallest upper limit.

**Algorithm 4.7:** Parallel Algorithm on Two Nodes

**Input:** An uncertainty graph as in Figure 4.3 (left), the number of allowed query rounds $r$, and a sequence of indices $0 = \ell_0, \ell_1, \ldots, \ell_r = m$.

**Output:** A feasible query set $Q$ after $r$ rounds.

```
Index all edges by increasing lower limit e_1, e_2, ..., e_m;
Initialize Q =: ∅ and i := 0;
while no MST is verified do
    Query all edges e_j with j ≤ ℓ_i that have not been queried and add them to the
      query set Q;
    Increase i by 1;
Return the query set Q;
```

Theorem 4.20 *Given an uncertainty graph on two nodes and $m$ edges and a bound $r \leq m/2$ on the number of query rounds, the algorithm* Parallel Algorithm on Two Nodes *with index sequence $0 = \ell_0, \ell_1, \ldots, \ell_r = m$ has competitive ratio*

$$\max_{i=1,\ldots,r} \left\{ \frac{m}{m-1}, \frac{\ell_{i+1}}{\ell_i + 1} \right\}.$$

*Proof.* The algorithm is correct, as it terminates either when an MST is verified, or after round $r$. In the latter case, all edges are queried and thus it is ensured that a minimum spanning tree can be identified.

By construction, the algorithm has queried $\ell_i$ edges after round $i$. To compare this with the optimal number of queries, we distinguish two cases: When the smallest edge weight is larger than $L_{e_m}$, all edges apart from the one with smallest upper limit must be queried to find the minimal edge. Then, any feasible query set contains at least $m-1$ edges and the algorithm queries at most $m$ edges. Otherwise, there exists an edge which has edge weight at most $L_{e_m}$. Then, to prove that an edge is minimal, all edges

whose interval overlaps with its edge weight need to be queried. This is the set of edges $e_1, \dots e_k$ up to a fixed index $k$. Thus, if the algorithm finishes in round $i$, the edges up to index $\ell_{i-1}$ do not suffice to verify an MST. Hence, any feasible query set has size at least $\ell_{i-1} + 1$. The algorithm queries $\ell_i$ edges. This means, the competitive ratio of the algorithm is

$$\max_{i=1,\dots,r} \left\{ \frac{m}{m-1}, \frac{\ell_i}{\ell_{i-1}+1} \right\}. \qquad \square$$

**Remark 4.21** For the sequence $0 = \ell_0, \ell_i = m^{i/r}$ for $i = 1, \dots, r$, Parallel Algorithm on Two Nodes has competitive ratio $\max\{m/(m-1), m^{1/r}\}$ for $m \geq 2$.

By duality, the dual algorithm has the same competitive ratio for the dual graph, which is a cycle $C_m$, as displayed in Figure 4.3 (right). We will use this dual algorithm in the next section as a subroutine and thus give its pseudo-code here for completeness.

**Algorithm 4.8:** Parallel Algorithm on a Cycle

**Input:** An uncertainty graph as in Figure 4.3 (right), the number of allowed query rounds $r$ and a sequence of indices $0 = \ell_0, \ell_1, \dots, \ell_r = m$.

**Output:** A feasible query set $Q$ after $r$ rounds.

```
Index all edges by decreasing upper limit e_1, e_2, ..., e_m;
Initialize Q := ∅ and i := 0;
while no MST is verified do
    Query all edges e_j with j ≤ ℓ_i that have not been queried and add them to the
     query set Q;
    Increase i by 1;
Return the query set Q;
```

Now we prove that Parallel Algorithm on Two Nodes is the best-possible algorithm for an uncertainty graph on two nodes with mutually overlapping intervals and describe the best-possible index sequence $\ell_0, \ell_1, \dots, \ell_r$.

**Theorem 4.22** *Given two positive integral constants $m \geq 2$ and $r \leq \lceil m/2 \rceil$. Any algorithm has competitive ratio in $\Omega(\alpha_1)$, where $\alpha_1$ is the second positive, real root of $\alpha(\alpha^r - (m+1)) + m$. Algorithm 4.8 attains this competitive ratio for the index sequence $\ell_i = (\alpha_1^{i+1} - \alpha_1)/(\alpha_1 - 1)$ for $1 \leq i < r$.*

*Proof.* We show the best sequence of $\ell_i$ for $i = 1, \ldots, r$ is choosing $\ell_i = \sum_{j=1}^{i} \alpha_1^j$ as defined in the theorem and then show this algorithm is best-possible on the instance. Let $\alpha$ be the competitive ratio of Parallel Algorithm on Two Nodes. As in the construction above, when the algorithm terminates it has queried $\ell_i$ edges and the optimal solution has size at least $\ell_{i-1} + 1$. Then, we must have $\ell_i/(\ell_{i-1} + 1) \leq \alpha$ for all indices $i = 1, \ldots, r$. This means we need $\ell_1 \leq \alpha, \ell_2 \leq \alpha^2 + \alpha, \ell_3 \leq \alpha^3 + \alpha^2 + \alpha$, which generalizes to $\ell_i \leq \sum_{j=1}^{i} \alpha^j$. At the same time, all $m$ edges need to be queried after $r$ rounds, which means $\ell_r \geq m$. Thus we get two equations for $\ell_r$:

$$\ell_r \geq m \qquad \ell_r \leq \sum_{j=1}^{r} \alpha^j = (\alpha^{r+1} - \alpha)/(\alpha - 1).$$

The smallest $\alpha$ which fulfills both equations is the second positive, real root of $\alpha(\alpha^r - (m + 1)) + m$.

To prove the Parallel Algorithm on Two Nodes with this index sequence is best-possible, we distinguish two cases. If some edge has weight at most $L_{e_m}$, then the order in which the edges are queried in Parallel Algorithm on Two Nodes is optimal. We choose the best-possible distribution of indices with the index sequence above, as then all inequalities are fulfilled with equality. Otherwise the smallest edge weight is larger than $L_{e_m}$. Then all edges apart from $e_1$ must be queried to find the minimal edge. Thus any feasible query set contains at least $m - 1$ edges and Parallel Algorithm on Two Nodes queries at most $m$ edges. However, $m/(m-1)$ is less than the bound $\alpha$ if and only if $r > log(2)/log(m/(m-1))$. For $m \geq 2$ and $r \leq m/2$, this is never the case. □

## Improving the Algorithm

On the lower bound example, we can improve upon our general algorithm performance $m/r$ by a lot. This raises the question if there is a better, general algorithm. We give such an algorithm under the restriction that the instance fulfills the preprocessing condition $T_L = T_U$. As before, we employ the main ideas of the algorithm Framework. However, we query four edges in each round, instead of two and still prove that at least two of these edges occur in any feasible query set. The key observation is, that either both edges chosen in the current cycle are in any feasible query set, or it is clear which is the next cycle that will be closed. This allows us to choose such an edge set of four edges almost always. Alternatively, there can be a second pair of edges for which at least one edge is in any feasible query set independent of the edge weights of the other edges, which we call *independent witness set*. This also allows us to query four edges. Last,

we show that if neither of these two constructions exists, then the uncertainty graph essentially is one of the two graphs in Figure 4.3. This in turn means we can apply Algorithm 4.7 or Algorithm 4.8 as our query strategy.

We call our algorithm Parallel MST (Algorithm 4.9). As in the algorithm Framework, we start with a lower limit tree $T_L$ and let this be our temporary edge set $\Gamma$. Iteratively, we add the other edges ordered by increasing lower limit.

Definition 4.23 *For an edge $h \in E \setminus T_L$, we consider the set of edges that have not occurred in any cycle before edge $h$ is added to $\Gamma$ and lie in the cycle which edge $h$ closes. If the uncertainty interval of such an edge $j$ overlaps with that of edge $h$, the two form an* independent witness set.

Whenever a cycle $C$ is closed in $\Gamma$ in iteration $i$, we consider the two edges $f_1, g_1 \in C$ with largest upper limits in Parallel MST. By Lemma 1.13 from Section 1.3, any feasible query set contains one of them. We query them together with two additional edges. If there is an independent witness set, we query these two edges additionally. Otherwise, we consider the edge $e_{i+1}$ that will be added next to $\Gamma$. We try to find an edge $g_2$ either in the cycle $e_{i+1}$ closes if $f_1$ is deleted or in the cycle $C$, such that at least two edges from $\{f_1, e_{i+1}, g_1, g_2\}$ occur in any feasible query set. If no such edge exists, we contract or delete all edges which are definitely in or not in an MST and apply one of the two algorithms Parallel Algorithm on Two Nodes or Parallel Algorithm on a Cycle. To count the number of times we query four edges, we use the variable $k$.

To prove the algorithm performance, we first show that for an independent witness set, any feasible query set contains at least one of the two edges.

Lemma 4.24 *Any feasible query set contains at least one of the edges in an independent witness set.*

*Proof.* Let the independent witness set contain the two edges $\{h, j\}$ and let $h$ be the edge that is not in $T_L$. We consider the cycle which edge $h$ closes with $\Gamma$. As edge $j$ does not occur in any cycle before edge $h$ is added to $\Gamma$, it is on the cycle which $h$ closes, independent of which edges are deleted before edge $h$ is added. By Lemma 1.14 from Chapter 1, the assumption $T_L = T_U$ yields that edge $h$, the edge which is newly added to $\Gamma$, has the largest upper limit in the cycle it closes. Thus, as edge $j$ will be in the same cycle independent of the previous deleted edges and $j$'s uncertainty interval overlaps that of edge $h$, at least one of the two edges is in any feasible query set by Corollary 1.15. □

**Algorithm 4.9:** Parallel MST

**Input:** An uncertainty graph $G = (V, E)$ and the max. number of query rounds $r$.

**Output:** A feasible query set $Q$ after $r$ rounds.

Determine a lower limit tree $T_L$ and set the temporary graph $\Gamma$ to $T_L$;
Index all edges in $E \setminus T_L$ by increasing lower limit $e_1, e_2, \ldots, e_{m-n+1}$;
Initialize $Q := \emptyset$ and $k := 0$;
**for** $i = 1$ *to* $m - n + 1$ **do**
    Add edge $e_i$ to the temporary graph $\Gamma$ and let $C$ be the occurring cycle;
    **while** *$C$ does not contain a maximal edge $e$ and $k < r$* **do**
        Choose $f_1 \in C$ such that $U_{f_1} = \max\{U_e | e \in C\}$;
        Choose $g_1 \in C \setminus \{f_1\}$ such that $U_{g_1} = \max\{U_e | e \in C \setminus f_1\}$;
        **if** *any following cycle contains an independent witness set* $\{h, j\}$ **then**
            Add $f_1, g_1, h, j$ to the query set $Q$ and query them;
        **else if** *there exists* $g_2 \in C \setminus \{f_1, g_1\}$ *with* $U_{g_2} > \max\{L_{f_1}, L_{g_1}\}$ *such that* $\{e_{i+1}, g_1\}$ *or* $\{e_{i+1}, g_2\}$ *is a witness set in the cycle closed by edge* $e_{i+1}$, *if edge* $f_1$ *is deleted from* $C$ **then**
            Add $f_1, e_{i+1}, g_1, g_2$ to the query set $Q$ and query them;
        **else**
            **if** $U_{g_1} < L_{e_{i+1}}$ **then**
                Apply the algorithm Parallel Algorithm on a Cycle on the cycle $C$ with index sequence $\ell_1 = 2, \ell_i = 2 + 4(i - 1)$ for $i = 1, \ldots, r - k - 1$ and $\ell_{r-k} = |C|$;
                Return the query set $Q$;
            **else**
                Contract all edges in $C \setminus \{f_1, g_1\}$, add the remaining edges to $\Gamma$ and apply the algorithm Parallel Algorithm on Two Nodes on the remaining graph with index sequence $\ell_1 = 2, \ell_i = 2 + 4(i - 1)$ for $i = 1, \ldots, r - k - 1$ and $\ell_{r-k} = 3 + m - n - i$;
                Return the query set $Q$;
        Increase $k$ by 1;
    Delete the edge with largest upper from $\Gamma$;
**if** $k = r$ **then**
    Query all edges In $E$ that have not been queried and add them to $Q$;
Return the query set $Q$;

Thus, if the algorithm queries edges in Algorithm Line 10, any feasible query set contains at least two of the four edges the algorithm queries.

If no independent witness set can be found in an iteration $i$, we use the following insight: If edge $f_1$ is not deleted from the cycle, but some other edge, then both edges $f_1$ and $g_1$ occur in any feasible query set. Otherwise, consider the cycle $C'$ which is closed by the next edge $e_{i+1}$, if $f_1$ is deleted from $C$.

If there is an edge $g_2 \in C \setminus \{f_1, g_1\}$ such that either the uncertainty intervals of edges $e_{i+1}, g_2$ overlap and $g_2$ is on $C'$, or the uncertainty intervals of both, $f_1, g_2$ and of $e_{i+1}, g_1$, overlap and $g_1$ is on $C'$, we query these four edges. To capture both cases, we call the two edges $\{g_1, g_2\}$ by $\{g, h\}$, without specifying which edge is which. We show that at least two of the edges $f_1, e_{i+1}, g, h$ occur in any feasible query set.

Lemma 4.25 *Consider a cycle $C$ in iteration $i$ of the algorithm together with a pair of edges $\{f_1, g\}$ in $C$, where edge $f_1$ has the largest upper limit in the cycle $C$ and the interval of edge $g$ overlaps that of edge $f_1$. Let $e_{i+1}$ be the next edge added to $\Gamma$ and let $h \in C \setminus \{f_1, g\}$ be an edge with overlapping uncertainty interval in the next cycle, if edge $f_1$ is deleted from the cycle $C$. Furthermore, let either $g$ or $h$ achieve $\max_{e \in C \setminus f_1} U_e$ and assume the intervals of $h$ and $g$ overlap. If $|\{f_1, e_{i+1}, g, h\}| = 4$ holds, then an optimal query set contains at least two of these four edges.*

*Proof.* By Lemma 1.13 at least one of the edges $f_1, g$ is in any feasible query set. We distinguish if edge $f_1$ is maximal in the cycle $C$ or not. If edge $f_1$ is maximal, then at least one of the two edges $f_2, h$ is in any feasible query set by Lemma 1.13. As $|\{f_1, f_2, g, h\}| = 4$, the two edge pairs are disjoint and thus any feasible query set contains at least one edge from each pair.

If edge $f_1$ is not maximal in the cycle $C$, it is in any feasible query set by Observation 1.2. The next edge with largest upper limit in $C$ is either edge $g$ or edge $h$ and it's uncertainty interval overlaps that of the other edge. Then, at least one of these two edges is in any feasible query set by Lemma 1.13. □

Theorem 4.26 *Given an uncertainty graph $G$ for which we cannot identify a minimum spanning tree. If no set of four edges can be found that fulfills the conditions of Lemmas 4.24 and 4.25, then we can contract and delete edges until one of the two graphs in Figure 4.3 remains.*

*Proof.* If we cannot identify a minimum spanning tree in the graph $G$, then the algorithm Parallel MST finds an edge pair $\{f_1, g_1\}$ in a cycle $C$ where no edge is maximal, such

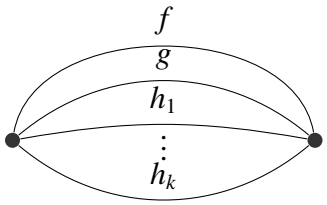



Figure 4.4: Graph obtained from $G$ after contracting edges in the MST.

that $f_1$ has the largest upper limit in $C$ and $g_1$'s uncertainty interval overlaps that of edge $f_1$. We denoted the edges that have not yet been added to $\Gamma$ by $h_1, ..., h_k$ and maintain their ordering such that $L_{h_1} \leq L_{h_2} \leq ... \leq L_{h_k}$. As edge $f_1$ is added to $\Gamma$ before $h_1$, $L_{f_1} \leq L_{h_1}$ holds. Furthermore, either $U_{g_1} \leq L_{h_1}$ holds or all edges $e \in C \setminus \{f_1, g_1\}$ have upper limit at most $L_{f_1}$.

By assumption, there is no edge pair fulfilling the conditions of Lemma 4.24. Thus, all edges not in the cycle $C$ apart from the edges $h_1, \ldots, h_k$ either do not occur in any cycle or their upper limit is smaller than the lower limit of the first edge $h_i$ closing a cycle they appear in. This means these edges are definitely in an MST, as for $j > i$ the lower limits of $h_j$. For simplicity, we contract them. This turns all edges $h_i$ into chords of the cycle $C$.

By assumption there is also no edge fulfilling the conditions of Lemma 4.25 with the edges $f_1, g_1$ and $h_1 = e_{i+1}$. Thus, all edges other than $g_1$ in the cycle $C$ must have an upper limit so small that their uncertainty interval does not overlap with that of edge $h_1$. Furthermore, either $g_1$'s interval does not overlap with $h_1$'s uncertainty interval, or all other cycle edges have upper limit at most $L_{f_1}$. In the first case, once the maximal edge in $C$ is deleted, all other edges have upper limit at most $U_{g_1} < L_{h_1}$. Thus, all edges $h_1, \ldots, h_k$ are maximal in their cycle and thus can be deleted from the graph. The resulting graph is the cycle graph from Figure 4.3 (right). In the other case, all edges apart from $f_1$ and $g_1$ on the cycle do not overlap with $f_1$'s interval. Consequently, they are definitely in the minimum spanning tree and can be contracted. The left over graph is the parallel graph in Figure 4.3 (left). It consists of only two nodes joint by the edges $f_1, g_1, h_1, ..., h_k$. □

For the graphs displayed in Figure 4.3, the two algorithms Parallel Algorithm on Two Nodes and Parallel Algorithm on a Cycle find a minimum spanning tree in few rounds.

This concludes the prerequisites to prove the algorithm performance.

Theorem 4.27 *Given an uncertainty graph and a number of allowed rounds r, Algorithm 4.9 has competitive ratio at most* $\max\{2, m/(2r-1)\}$.

*Proof.* If the algorithm always executes Algorithm Line 10 or Algorithm Line 12 and uses $k < r$ rounds, in every iteration exactly four edges are queried. Here, the optimal solution has at least $2k$ edges by Lemmas 4.24 and 4.25, which yields a competitive ratio of 2. If the algorithm uses $r$ rounds, at most $m$ edges are queried and the optimal solution has size at least $2r$. However, if the algorithm turns to Algorithm Line 18 after some rounds, then for one round we query only two edges from which at least one is in any feasible query set by Theorem 4.20. In the next rounds we query four edges again, out of which at least two are in any feasible query set by the same theorem. Thus, if the algorithm terminates in round $k < r$ it has competitive ratio 2. If it takes $r$ rounds, an optimal query set has size at least $2r - 1$ and the algorithm queries at most $m$ edges. □

### 4.5.2 Parallelization of k-th smallest value

We consider the $k$-th smallest value problem again. Given a set of $n$ elements, each with uncertainty interval $A_i$, we aim to find the $k$-th smallest of these elements by querying as few elements as possible for their exact value. In the parallelization model, we are given a fixed maximum number of $r$ rounds to find the $k$-th smallest element. As before, we assume $k \leq n/2$ as all results transfer to larger $k$ by symmetry.

For a lower bound depending on the number of rounds $r$, we observe that finding the largest value is actually exactly the problem of finding a minimum spanning tree on a cycle, e.g. finding the largest weight edge on a cycle. As finding the $k$-th smallest value of an instance with $k-1$ intervals smaller than all others is equivalent to finding the smallest value of the rest of the set, we also have a lower bound $\lceil (n-k+1)^{1/r} \rceil$ on the competitive ratio. Combined, this yields the following lower bounds:

Lemma 4.28 *Given r rounds and a set of n elements with uncertain weight, no algorithm that computes the k-th smallest value of the set in r rounds has competitive ratio* $c < \max\{2, k, \lceil (n-k+1)^{1/r} \rceil\}$.

We first consider the special case $k = 1$. Finding the smallest value of a set is the same as finding the minimum spanning tree of a graph with two nodes connected by parallel edges. Hence, we can apply our algorithm Parallel Algorithm on Two Nodes for this case, which is best-possible.

**Theorem 4.29** *There is a* $\max\{n/(n-1), \lceil n^{1/r} \rceil\}$*-competitive algorithm for the smallest/largest value problem in* $r$ *rounds and this is best-possible.*

If we consider an additive performance guarantee for Parallel Algorithm on Two Nodes, we choose index sequence $\ell_i = i \cdot n/r$. Then, if the algorithm terminates in round $i$ it queries $\ell_i$ elements. At the same time an optimal solution contains at least $\ell_{i-1} + 1$ elements. Thus, the performance guarantee is $OPT + n/r - 1$ for any $i$, which means the index sequence is best-possible.

**Theorem 4.30** *There is an algorithm that finds the smallest/largest value of a set of elements with uncertain weight, which uses at most* $OPT + \max\{1, \lceil n/r \rceil - 1\}$ *queries and this is best-possible.*

For $k > 1$, we give an algorithm based on the same idea as Parallel Algorithm on Two Nodes. Parallel $k$-th Smallest Value. Our algorithm Algorithm 4.10, sorts all elements by lower limits and queries the elements in this order. In each round $j$, we query the elements up to index $\ell_j$, according to a sequence of indices $\ell_j$ that are given as the input. As soon as we can identify the $k$-th smallest element at the end of a round, the algorithm terminates.

**Algorithm 4.10:** Parallel $k$-th Smallest Value

**Input:** A set of elements $X = \{e_1, ..., e_n\}$ and a sequence of indices
$0 = \ell_0 \leq \ell_1 \leq \cdots \leq \ell_r = n$.

**Output:** A feasible query set $Q$.

```
1 Initialize Q := ∅, j := 1;
2 Sort the elements by lower limits and reindex them such
  that L_{e_1} ≤ L_{e_2} ≤ ··· ≤ L_{e_n};
3 while no k-th smallest element is identifiable do
4     Set Q = {e_1, ..., e_{ℓ_j}} and query the new elements;
5     Increase j by 1;
6 Return the query set Q;
```

**Theorem 4.31** *Given* $r$ *rounds, a set of* $n$ *elements of uncertain weight and an index* $k \geq 1$*. Let* $\alpha_1$ *equal the first real root of* $(\alpha^{r+1} - k(\alpha^r - \alpha) - \alpha n - \alpha) + n$ *that is at least* $k$*. Then Algorithm 4.10 with* $\ell_j = (\alpha_1^{j+1} - k(\alpha_1^j - \alpha_1) - \alpha_1)/(\alpha_1 - 1)$ *computes the*

*k-th smallest value with multiplicative competitive ratio $\alpha_1$ and this is the best-possible choice of $\ell_j$. For $k = 1$ we have $\alpha_1 = n^{1/r}$.*

*Proof.* Let $Q^*$ be the optimal query set and $Q$ be the query set of the algorithm. Let $i+1$ be the round after which the algorithm finishes and denote with $Q_i$ the query set of the algorithm at the end of round $i$. Let $x$ be the element of $k$-th smallest value. In Algorithm 4.10 we query elements ordered by increasing lower limit. Thus we always query the unqueried elements with smallest lower limit next. If all elements with $L_e \leq w_x$ have been queried, we can identify the $k$-th smallest element. This means we have $L_e \leq w_x$ for all elements in $Q_i$. In Section 4.3.1 we have already considered the $k$-th smallest value problem. Similar to Lemma 4.4, also for this algorithm holds $|Q_i| \leq |Q^* \cap Q_i| + k$. This is because any element $e \neq x$ with $w_x \in (L_e, U_e)$ must be in any feasible query set and there are at most $k-1$ elements with smaller upper limit than $w_x$. The algorithm starts round $i+1$ and thus at least one element from $X \setminus Q_i$ must be in the optimal query set. Consequently $|Q_i| \leq |Q^*| + k - 1$. Let $\alpha$ be the competitive ratio of Algorithm 4.10. Then we need $|Q| = \ell_{i+1} \leq \alpha \cdot |Q^*|$ for any $0 \leq i < r$, as the algorithm can terminate in any round $i+1$. Using our bound on the size of $|Q^*|$, this means we need

$$|Q| = \ell_{i+1} \leq \alpha \cdot |Q^*| \leq \alpha \cdot (\ell_i - k + 1) \leq \alpha^{i+1} - (k-1) \sum_{t=1}^{i} \alpha^t = \frac{\alpha^{i+2} - k(\alpha^{i+1} - \alpha) - \alpha}{\alpha - 1}.$$

The algorithm must finish after at most $r$ rounds and in the worst case all $n$ elements must queried. Hence we need

$$\ell_r \geq n \qquad \text{and} \qquad \ell_r \leq \frac{\alpha^{r+1} - k(\alpha^r - \alpha) - \alpha}{\alpha - 1}.$$

The root $\alpha_1$ is the smallest value fulfilling both of these inequalities. □

To attain an algorithm with good additive performance we use a different index function $\ell_j$.

**Theorem 4.32** *Given $r$ rounds, a set of $n$ elements of uncertain weight and an index $k \geq 1$, Algorithm 4.10 with $\ell_j = j \cdot \max\{k, \lceil n/r \rceil\}$ computes the $k$-th smallest value using at most OPT$+k-1+\max\{k, \lceil n/r \rceil\}$ queries.*

*Proof.* Let $Q^*$ be the optimal query set and $Q$ be the query set of the algorithm. Let $i$ be the round after which the algorithm finishes and for each round $j < i$ denote with $Q_j$ the query set of the algorithm at the end of round $j$. Let $x$ be the element of $k$-th smallest value. By the same argumentation as in the proof of Theorem 4.31, we have $|Q_j| \leq$

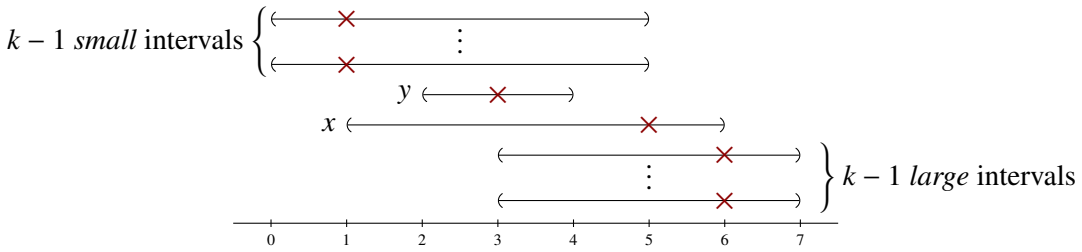



Figure 4.5: Example that Theorem 4.32 is almost tight.

$|Q^*| + k - 1$ for $j < i$. For each round $j \geq 1$ the algorithm's query set $Q_j$ has size at most $j \cdot \max\{k, \lceil n/r \rceil\}$. Thus we have for $i \geq 2$ for the algorithm query set $Q$

$$|Q| = |Q_{i-1}| + \max\{k, \lceil n/r \rceil\} \leq |Q^*| + k - 1 + \max\{k, \lceil n/r \rceil\}.$$

For $i = 1$ we have $|Q| = \max\{k, \lceil n/r \rceil\} \leq \max\{k, \lceil n/r \rceil\} \cdot |Q^*|$. □

In Figure 4.5 we display an example that proves that this analysis is almost tight for Algorithm 4.10. Consider $2k$ elements, where $k-1$ elements have interval $(0, 5)$, another $k - 1$ elements have interval $(3, 7)$ and there are two additional elements $x$ with interval $(1, 6)$ and $y$ with interval $(2, 4)$. Let the realization give weight 1 to all small elements, weight 6 to all large elements and weights $w_x = 5$ and $w_y = 3$ to the other two elements. Then Algorithm 4.10 first queries the $k$ elements with smallest lower limit. These are the $k - 1$ small elements as well as the element $x$. Once their weight is revealed, the $k$-th smallest element can not be identified. It could be any of the $k$ elements that have not been queried yet. In the next round, the algorithm queries these $k$ elements and then identifies element $y$ as the $k$-th smallest element. Thus, the algorithm needs $2k$ queries to find the $k$-th smallest element. However, it suffices to query the two elements $x$ and $y$ to prove that $y$ is the $k$-th smallest element. Hence, the optimal query set has size 2 and algorithm performance is $OPT + 2k - 2$.

### 4.5.3 Sequencing

For *sequencing under uncertainty* our algorithm Overlap Vertex Cover needs only 2 rounds. Thus, there is no trade-off between algorithm performance and number of rounds for $r \geq 2$. For a single round we show the competitive ratio can get arbitrarily large. Consider one element $e_0$ with uncertainty interval $A_0 = (0, n + 1)$ and $n$ additional elements $e_1, ..., e_n$ with uncertainty interval $A_i = (i, i + 1)$ for $i = 1, ..., n$. Given just one

round, an algorithm must query all intervals to ensure after this round the ordering is unique. However, for example for the realization where $w_0 = 1$, the optimal query set contains just element $e_0$. Thus the competitive ratio is $n + 1$ here.

Theorem 4.33 *Given a set of $n$ elements of uncertain weight and a round number $r$,* Overlap Vertex Cover, *Algorithm 4.3, has competitive ratio 2 if $r \geq 2$ and for $r = 1$ it has competitive ratio $n$. In both cases it is best-possible.*

We can find the set of elements we need to query if we only allow 1 round in polynomial time, as these are all elements with non-zero degree in the overlap graph. This is also the set of all elements, that might potentially be in the optimal query set.

## 4.6 Alternative Query Types

We consider the extension of our query model, in which a query to an edge may return an open subinterval of the current uncertainty interval instead of a point. In this model, several queries to one edge might be necessary. The model was first analyzed in [GSS16] under the name OP-OP model, meaning the original uncertainty intervals are open intervals or points and the query output as well. They show for *MST under uncertainty* that the deterministic 2-competitive algorithm by Erlebach et al. [EHK+08] extends to the OP-OP model without any loss in the competitive ratio. We analyze the OP-OP model for randomized algorithms and show the surprising fact, that no improvement over competitive ratio 2 is possible using randomization. For non-uniform query cost, we show that we cannot use the algorithm Balance presented in Section 1.5.2. We give a new algorithm that attains competitive ratio 2, proving that there is no loss in the algorithm performance from the uniform query cost model allowing only points to the non-uniform query cost model with intervals.

### 4.6.1 Randomized Algorithms

We consider a randomized problem instance $(G, \mathcal{R}, p)$, that is an uncertainty graph $G$ together with a family of feasible realizations $\mathcal{R}$ and a probability distribution $p$ on these realizations. We use $R \sim_p \mathcal{R}$ to denote a realization $R$ drawn from $\mathcal{R}$ according to $p$. For such a randomized instance, we show that for no deterministic algorithm the expected

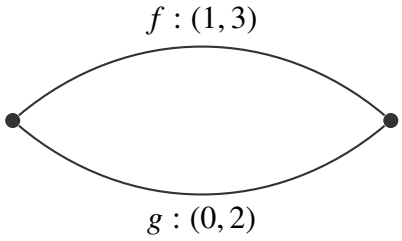



Figure 4.6: Lower bound example for OP-OP randomized

ratio of $ALG/OPT$ is less than 2. Applying a variant of Yao's Principle [Yao77,BEY98], this yields that no randomized algorithm has competitive ratio smaller than 2.

Theorem 4.34 (Variant of Yao's Principle [BEY98, Thm. 8.5]) *Let $\mathcal{A}$ denote the class of all deterministic algorithms and let $\mathcal{F}$ be the family of all randomized instances $(G, \mathcal{R}, p)$ for a minimization problem. Then any randomized algorithm has a competitive ratio $c$ for which holds*

$$c \geq \min_{ALG \in \mathcal{A}} \mathbb{E}_{R \sim_p \mathcal{R}} \left[ \frac{ALG(G,R)}{OPT(G,R)} \right] \qquad \forall (G, \mathcal{R}, p) \in \mathcal{F}.$$

Consider the uncertainty graph depicted in Figure 4.6, with two edges $f$ and $g$ joining the same pair of vertices and uncertainty intervals $A_f = (1, 3)$ and $A_g = (0, 2)$. For a fixed parameter $n \in \mathbb{Z}_{>0}$ we first define the family $\mathcal{R}_n$ of $2n$ feasible realizations and then give a probability distribution on them. Let realization $R_j$ reveal for $j-1$ queries to edge $f$ uncertainty interval $(1, 3)$ and for the $j$-th query interval $[2, 2]$. Here the uncertainty interval of edge $g$ stays $(0, 2)$ for $n$ queries and turns to the trivial uncertainty interval $[1, 1]$ upon the $n$ + 1-st query. Symmetrically let realization $R_{-j}$ reveal edge weight $[1, 1]$ upon the $j$-th query to edge $g$ and edge weight $[2, 2]$ with the $n+1$-st query to edge $f$. Then the optimal strategies for realizations $R_j$ and $R_{-j}$ on $G$ are to query edge $f$ or respectively edge $g$ repeatedly for $j$ times and thus make $j$ queries in total.

We define a randomized instance $(G, \mathcal{R}_n, p)$ by giving a distribution $p$ over the realizations in $\mathcal{R}_n$. Let each of the $2n$ realizations $R_j$ and $R_{-j}$, $j = 1, \ldots, n$, occur with probability $P(R_j) = P(R_{-j}) = 1/2n$ in the distribution $p$.

Consider the algorithm ALG, that alternates between querying edge $f$ and edge $g$. If we denote by $f^i, g^i$ the $i$-th query to edges $f$ and $g$, the query sequence of the algorithm is: $f^1, g^1, f^2, g^2, \ldots, f^n, g^n$. We compute the competitive ratio of ALG and then show its performance is best-possible.

Lemma 4.35 *The Algorithm ALG defined above has competitive ratio at least* 2.

*Proof.* Consider the randomized instance $(G, \mathcal{R}_n, p)$ for $n \in \mathbb{Z}_{>0}$ defined above. We show algorithm ALG has competitive ratio 2 when $n$ tends to infinity. The algorithm ALG needs $2j-1$ queries when realization $R_j$ occurs, as it queries edge $f$ for the $j$-th time after querying both edges $j-1$ times. For realization $R_{-j}$ it needs $2j$ queries. The optimal query set has size $j$ for both realizations $R_j$ and $R_{-j}$. Thus the competitive ratio for the randomized instance $(G, \mathcal{R}_n, p)$ is:

$$\mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}(G,R)}{\mathrm{OPT}(G,R)}\right] = \sum_{j=1}^{n} P(R_j)\frac{2j-1}{j} + \sum_{j=1}^{n} P(R_{-j})\frac{2j}{j} = 2 - \frac{1}{2n}\sum_{j=1}^{n}\frac{1}{j}.$$

The sum expresses the harmonic number $H_n$, which has growth less than $1/n$, and thus we get

$$\mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}(G,R)}{\mathrm{OPT}(G,R)}\right] = 2 - \frac{1}{2n}\cdot H_n \overset{n\to\infty}{\longrightarrow} 2.$$

This proves Algorithm ALG has competitive ratio at least 2 on the randomized instance $(G, \mathcal{R}_n, p)$ and thus also in general. □

Lemma 4.36 *No algorithm performs better than ALG on the randomized problem instance $\mathcal{R}_n$ with probability distribution $p$.*

*Proof.* We observe first, that any algorithm obeying the principle that one edge is queried for the $i$-th time only after the other edge has been queried for $i-1$ times has the same competitive ratio as ALG, as the family of realizations $\mathcal{R}_n$ and the probability distribution $p$ are symmetric in $f$ and $g$.

Now consider an algorithm $\mathrm{ALG}_1$ not obeying this principle. Its query sequence contains edges $f$ and $g$ each $n$ times in an arbitrary order. By definition it has a point in the query sequence, where the number of queries to edge $f$ and to edge $g$ differs by at least 2. Then the query sequence also contains two consecutive queries whose query numbers differ by at least 2. Without loss of generality assume their order is $g^y, f^x$ and $y \geq x+2$ for two integers $x$ and $y$. We define a new algorithm $\mathrm{ALG}_2$ and show that it has strictly smaller competitive ratio. Let $\mathrm{ALG}_2$ be the algorithm where we switch these two queries $g^y, f^x$ and that thus contains the sequence $f^x, g^y$.

The number of queries of $\mathrm{ALG}_1$ and $\mathrm{ALG}_2$ coincides for all realizations $R \notin \{R_x, R_{-y}\}$. Using the linearity of expected values, this means the difference of the two competitive

ratios simplifies to

$$\mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}_2(G,R)}{\mathrm{OPT}(G,R)}\right]-\mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}_1(G,R)}{\mathrm{OPT}(G,R)}\right]$$
$$=\mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}_2(G,R)-\mathrm{ALG}_1(G,R)}{\mathrm{OPT}(G,R)}\right]=\frac{P(R_{-y})\cdot 1}{\mathrm{OPT}(G,R_{-y})}-\frac{P(R_x)\cdot 1}{\mathrm{OPT}(G,R_x)}.$$

The number of queries for realization $R_x$ is one larger for Algorithm $\mathrm{ALG}_1$ than for $\mathrm{ALG}_2$. For realization $R_{-y}$ it is the other way around. Hence, we get

$$\mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}_2(G,R)}{\mathrm{OPT}(G,R)}\right]-\mathbb{E}_{R\sim_p\mathcal{R}_n}\left[\frac{\mathrm{ALG}_1(G,R)}{\mathrm{OPT}(G,R)}\right]=\frac{1}{2n}\left(\frac{1}{y}-\frac{1}{x}\right)=\frac{x-y}{2nxy}<0.$$

This yields that the performance of $\mathrm{ALG}_2$ is strictly better than the performance of $\mathrm{ALG}_1$. Any algorithm that queries $f$ and $g$ alternatingly has competitive ratio 2 and any other algorithm is not best-possible. Thus algorithm ALG with competitive ratio 2 is best-possible for the randomized instance $(G,\mathcal{R}_n,p)$. □

We apply Yao's Principle (Theorem 4.34) to Lemma 4.36 to prove our claim.

**Theorem 4.37** *There is no randomized algorithm for* MST under uncertainty *in the OP-OP model with competitive ratio $c<2$.*

**Corollary 4.38** *The 2-competitive deterministic algorithm which is presented by Gupta et al. [GSS16] has best-possible competitive ratio, even among randomized algorithms.*

## 4.6.2 Non-uniform Query Cost

We first show that Balance is not necessarily 2-competitive with queries returning subintervals. Then we present a new algorithm with competitive ratio 2 that can handle more general queries. It carefully computes the query cost we can save with each element, if we do not query it.

We consider two triangles sharing one edge with three distinct edges $f,g,h$ as displayed in Figure 4.7. Balance starts with the lower limit tree $T_L$ containing the two edges of weight 1 and edge $f$. We add the other edges by increasing lower limit and thus add edge $g$ before edge $h$. This means the first closed cycle is $C$. Edges $f$ and $g$ will be queried in the algorithm until one of the two is maximal in the cycle. This means Balance makes 6 queries for the following interval sequences:

$$(1,9)\to(5,9)\to(7,9)\to[8,8]$$
$$(2,10)\to(6,10)\to(8,10)\to(9,10).$$

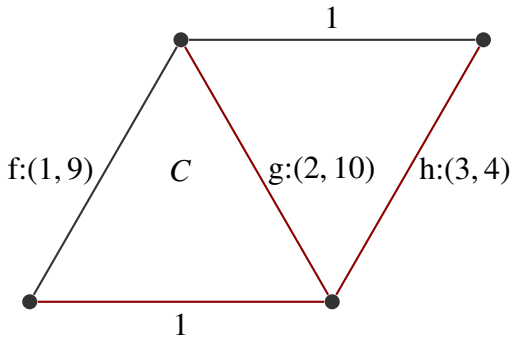



Figure 4.7: Example for Balance.

Once $g$ has been deleted, edge $h$ is added and the cycle with $f$ and $h$ occurs. Immediately, edge $f$ is maximal and we find a minimal spanning tree with the two edges of weight 1 and edge $h$. An optimal strategy would have queried edges $f$ and $g$ only once and then realized that they are dominated by edge $h$. Thus, two queries suffice to solve the instance. It is clear that the weights of edges $e$ and $f$ can be adapted such that for any competitiveness factor $k$ we have a counter example.

**Observation 4.39** Balance *is not constant factor competitive if intervals are allowed as query output.*

We present an alternative algorithm Interval Handler that computes a minimum spanning tree with non-uniform query costs and intervals as query output. It merges ideas from the algorithm U-red of Erlebach et al. [EHK+08], our algorithm Balance presented in Section 1.5, and the verification algorithm from [EH14]. Algorithm 4.11, Interval Handler, starts with a lower limit tree $T_L$ and then iterates over the edges by increasing lower limit as before. However, in each cycle we consider the edge pair of the two edges with largest upper limit instead of the neighborhood. We introduce a value $v_e \in \mathbb{R}_{\geq 0}$ for every edge, that counts the query cost accumulated by edges in a pair with it. In a sense, the term $c_e - v_e$ characterizes the potential query cost we may save by not querying edge $e$. At first the potential saved query cost for each edge is its query cost. When the edge occurs in a cycle pair, the potential saved query cost decreases, as not querying the one edge means we have to query the other. All edges that contribute to $v_e$ must be queried if $e$ is not queried. For each edge pair in the algorithm, we decide to query the edge $e$ that has a smaller potential save of query costs if it is not queried. The edge $f$ that has the larger potential save in query cost, is not queried in this step. Then we increase the value of the edge $f$ we did not query by $c_e - v_e$. A key feature of

the algorithm, is that after an edge is queried its value is set back to 0 and the algorithm is restarted, but the value of the other edges is not reset.

**Algorithm 4.11:** Interval Handler

**Input:** An uncertainty graph $G = (V, E)$.

**Output:** A feasible query set $Q$.

1 Initialize the value $v_e$ at 0 for every edge $e \in E$ and set $Q := \emptyset$.;
2 Let $T_L$ be the lower limit tree and index remaining edges by increasing lower limit $e_1 \leq e_2 \leq \cdots \leq e_{m-n+1}$;
3 Let $\Gamma$ be $T_L$;
4 **for** $i = 1$ *to* $m - n + 1$ **do**
5  Add $e_i$ to $\Gamma$;
6  **while** $\Gamma$ *has a cycle* $C$ **do**
7  **if** $C$ *contains an always maximal edge* $e$ **then**
8  Delete $e$ from $\Gamma$;
9  **else**
10  Choose $f \in C$ s.t. $U_f = \max\{U_e | e \in C\}$;
11  Choose $g \in C \backslash f$ s.t. $U_g = \max\{U_e | e \in C \backslash f\}$;
12  **if** $A_g$ *is trivial* **then**
13  Query edge $f$, add it to $Q$ and set $v_f = 0$;
14  **else if** $c_f - v_f \geq c_g - v_g$ **then**
15  Query edge $g$, add it to $Q$, add $c_g - v_g$ to $v_f$, and set $v_g = 0$;
16  **else**
17  Query edge $f$, add it to $Q$, add $c_f - v_f$ to $v_g$, and set $v_f = 0$;
18  Restart the algorithm in Algorithm Line 2;
19 Return the query set $Q$;

Algorithm 4.11 outputs a minimal spanning tree, as after the last restart all edges have been considered only maximal edges in cycles have been deleted and a spanning tree remains. It terminates because in each iteration of the cycle loop in Algorithm Line 6 one cycle edge is queried or deleted and after a finite number of queries to each edge its interval reduces to a point. At the latest, when all edges in the graph have trivial uncertainty interval, we find an always maximal edge in each cycle.

We want to show that Interval Handler has competitive ratio 2. For this, we first observe that after each query to an edge its value is reset to 0. Then, the algorithm restarts, but maintains the value of all other edges. Thus, querying an edge is as if the edge is deleted and replaced by a parallel edge with an uncertainty interval that is either the same or contained in the old interval. This simplifies the instance in the sense than any feasible query set for the original uncertainty graph is also a feasible query set for the uncertainty graph after some edge is queried. We make the following observation concerning the restart of our algorithm Interval Handler.

Observation 4.40 *An edge that is in any feasible query set in some run $j$ of* Interval Handler*, is also in any feasible query set for the original graph $G$.*

Thus, for the proof of the algorithm performance, we treat multiple queries to one edge as if they would be queries to different, parallel edges. Interval Handler behaves exactly like Framework and thus we can apply Lemma 1.13 from Section 1.3 to each run of the algorithm. Together with Observation 4.40 we can formulate a corollary to Lemma 1.13.

Corollary 4.41 *Given an uncertainty graph $G$, then any feasible query set contains at least one of the two edges $f$ and $g$ chosen by* Interval Handler*. Furthermore, if $A_g$ is trivial, any feasible query set contains edge $f$.*

As in [EH14], we create a forest $H$ using this property. We create the graph solely for the purpose of the proof. This means we observe the algorithm behavior for a particular problem instance and build the graph depending on this. For each edge in the algorithm we create one node. Whenever an edge is queried, we create a second node for a possible second query of this edge. We add an edge between two nodes, when they occur as an edge pair in Algorithm Line 15 or Algorithm Line 17 of Algorithm 4.11. Let an edge that is queried in this step of the algorithm be the child of the edge it is paired up with at that time. The partner has not been queried, which means the graph is cycle-free.

By construction, any feasible query set contains at least one of the two vertices of every edge in $H$.

Observation 4.42 *Let $H$ be the graph defined as described above from a run of* Interval Handler*. Then the set of edges queried by an optimal solution contains a vertex cover of $H$.*

Let us consider a tree $T$ of this forest $H$. We show we can bound the query cost of the algorithm in $T$ by twice the query cost of the optimal solution restricted to the edges of $T$. We need the following notation: Let $T$ be the defined tree, then we denote with $T_e$ the subtree that has node $e$ as its root. If $r$ is the root of a tree $T$, then $T = T_r$ holds. We define $C(e)$ as the set of all children of $e$ and $GC(e)$ as the set of all grand children.

The tree is structured in such a way that there is a relation between the query cost and the values of the child and parent nodes. A child $a$ is queried when it appears in a pair with its parent $b$ in the algorithm. In Algorithm Line 15 or Algorithm Line 17, respectively, the term $c_a - v_a$ is added to the value $v_b$. In no other situation anything is added to its value. Therefore $v_b$ is the sum of the term $c_a - v_a$ over all children $a$ of $b$. Additionally we know that $c_e - v_e$ is non-negative in any state of the algorithm.

Observation 4.43 *For the tree $T$ and a node $e$ of the tree, it holds that $c_e \geq v_e = \sum_{f \in C(e)} (c_f - v_f)$.*

Any edge that is the child of another edge is queried in the algorithm. Thus, the tree $T$ has at most one edge that is not queried, its root. For all edges $b \in T$ that are queried, we consider their subtree $S_b$. We relate the optimal cost of this subtree $OPT(S_b)$ to the optimal cost $OPT(S_b | b \in OPT)$, if $b$ is required to be contained in $OPT$.

Lemma 4.44 *For an edge $b$ that is queried in Algorithm Line 15 or Algorithm Line 17 of* Interval Handler *and its set of related edges $S_b$ holds*

$$OPT(S_b | b \in OPT) = c_b - v_b + OPT(S_b).$$

*Proof.* We use induction on the number of generations contained in $S_b$. If the set $S_b$ contains only edge $b$, the value of edge $b$ is zero. This means the optimal query set for $S_b$ is either empty, or $b$, if edge $b$ is required to be included in the optimal solution. Hence,

$$OPT(S_b | b \in OPT) = c_b = c_b - 0 + 0 = c_b - v_b + OPT(S_b).$$

For an edge $b$ with non-empty children set $C(e)$, the optimal solution containing edge $b$ has cost $c_b$ plus the optimal solution of the subtrees which have $b$'s children as their root. Applying induction and Observation 4.43 for $v_b$, this yields

$$\begin{aligned} OPT(S_b) &= c_b + \sum_{e \in C(b)} OPT(S_e) = c_b - \sum_{e \in C(b)} c_e - v_e + \sum_{e \in C(b)} OPT(S_e) \\ &= c_b - v_b + OPT(S_b | b \notin OPT). \end{aligned}$$

□

For any edge $a \in T$ we apply Lemma 4.44 to its children, to get the following corollary:

**Corollary 4.45** *For an edge $b \in T$ of* Interval Handler *and its set of related edges $S_b$ holds*

$$OPT(S_b) = \sum_{e \in C(b)} c_e + \sum_{e \in GC(b)} OPT(S_e).$$

If the root of $T$ is not queried in the algorithm, the algorithm cost is the sum of the query cost of all other edges in the tree. We show the optimal query cost restricted to $T$ is at most twice as large.

**Lemma 4.46** *If $T$ contains more than one edge of the uncertainty graph, for any edge $a \in T$ and its set of related edges $S_a$, then holds*

$$2 \cdot OPT(S_a) = v_a - c_a + \sum_{e \in S_a} c_e.$$

*Proof.* We use induction on the number of generations contained in the set of related elements $S_a$. If edge $a$ does not have any grandchildren, we apply Corollary 4.45 and Observation 4.43 to obtain

$$2 \cdot OPT(S_a) = 2 \cdot \sum_{e \in C(a)} c_e = \sum_{e \in C(a)} c_e + \sum_{e \in C(a)} c_e - v_e = \sum_{e \in S_a} c_e - c_a + v_a.$$

For an edge $a$ with non-empty grandchildren set we also apply Corollary 4.45. Then we apply induction to get

$$\begin{aligned} 2 \cdot OPT(S_a) &= 2 \cdot \sum_{e \in C(a)} c_e + \sum_{e \in GC(a)} 2 \cdot OPT(S_e) \\ &= 2 \cdot \sum_{e \in C(a)} c_e + \sum_{e \in GC(a)} \left[ v_e - c_e + \sum_{h \in S_e} c_h \right]. \end{aligned}$$

By repeated application of Observation 4.43 this yields

$$2 \cdot OPT(S_a) = \sum_{e \in C(a)} c_e - v_e + \sum_{e \in S_a} c_e - c_a = v_a - c_a + \sum_{e \in S_a} c_e.$$ □

If the algorithm queries the root of the tree $T$, it is queried in Algorithm Line 13 of the algorithm. Then, by Corollary 4.41, the optimal solution also contains the edge corresponding to the root.

Lemma 4.47 *For an edge $a$ that is queried in Algorithm Line 13 of* Interval Handler *and its set of related edges $S_a$ holds*

$$2 \cdot OPT(S_a) \geq \sum_{e \in S_a} c_e.$$

*Proof.* We use induction on the number of generations in the set $S_a$. If edge $a$ does not have any children, the optimal solution is equal to the sum of the query cost of all edges in the set $S_a$. If the set of the children of edge $a$ is non-empty, we apply Corollary 4.45 and use the fact that the value of an edge does not exceed the edge cost (Observation 4.43) to obtain

$$\begin{aligned} 2 \cdot OPT(S_a) &= 2 \cdot c_a + 2 \cdot \sum_{e \in C(a)} OPT(S_e) \geq c_a + v_a + 2 \cdot \sum_{e \in C(a)} OPT(S_e) \\ &= c_b + \sum_{e \in C(a)} \left[ c_e - v_e + OPT(S_e) \right]. \end{aligned}$$

For the children of $a$, Lemma 4.46 holds. Thus, we can conclude

$$2 \cdot OPT(S_a) = c_a + \sum_{e \in C(a)} \sum_{h \in S_e} c_f = \sum_{e \in S_a} c_e.$$ □

This completes all preliminaries to prove the algorithm performance.

Theorem 4.48 Interval Handler, *Algorithm 4.11, achieves competitive ratio* 2 *in the OP-OP model where queries may return intervals.*

*Proof.* We consider the forest $H$ that is defined by a run of the algorithm. It partitions the edges into a family of trees. For each tree $T$ where the algorithm does not query the edge $a$ corresponding to the root node, its query cost is the sum of the query costs $c_e$ for all $e \in T \setminus a$. By Lemma 4.46 this is at most twice the optimal cost restricted to $T$. For each tree $T$ in which the root $a$ is queried by the algorithm, Lemma 4.47 proves that the algorithm query cost does not exceed twice the optimal query cost restricted to this edge set. As the trees of the partition are disjoint, this concludes the proof. □

Remark 4.49 A factor 2 for the competitive ratio is best-possible for deterministic algorithms, as it is a lower bound for the uniform query cost case.

Chapter 5

# An Adversarial Model for Scheduling with Testing

In this chapter, we consider a novel model for scheduling with explorable uncertainty. In this model the processing time of a job can potentially be reduced (by an *a priori* unknown amount) by testing the job. Testing a job $j$ takes one unit of time and may reduce its processing time from the given upper limit $\bar{p}_j$ (which is the time taken to execute the job if it is not tested) to any value between 0 and $\bar{p}_j$. This setting is motivated e.g. by applications where a code optimizer can be run on a job before executing it. We consider the objective of minimizing the sum of completion times. All jobs are available from the start, but the reduction in their processing times as a result of testing is unknown, making this an online problem that is amenable to competitive analysis. The need to balance the time spent on tests and the time spent on job executions adds a novel flavor to the problem.

We give first and nearly tight lower and upper bounds on the competitive ratio for deterministic and randomized algorithms. We also show that minimizing the makespan is a considerably easier problem for which we give optimal deterministic and randomized online algorithms.

Remark: The results in this chapter are based on joint work with Christoph Dürr, Thomas Erlebach, and Nicole Megow, published at *Innovations in Theoretical Computer Science 2018* [DEMM18]. A full version is maintained at arXiv [DEMM17].

Previously, in the area of explorable uncertainty the execution of queries was assessed separately from the actual optimization problem being solved, yielding a bi-criteria optimization problem. However, when queries as the optimization goal use the same resource like time, cost, or energy and thus affect each other, there is a joint objective function. This is a new direction for optimization with explorable uncertainty applicable to all areas where uncertainty exploration and solution compete for the same resource.

Our model scheduling with testing is a first example of uncertainty exploration with a single objective function. We consider single machine scheduling with $n$ jobs. Every job can optionally be *tested* prior to its execution. A job that is executed without testing has processing time $\bar{p} \in \mathbb{Q}^+$, while a tested job has some processing time in $[0, \bar{p}]$ that is initially unknown. It takes unit time to test a job, which reveals its processing time. Tested jobs can be executed at any time after their test. Unless otherwise noted, we consider the sum of completion times as the minimization objective. Testing a job gains information and possibly reduces the processing time of a job, but it may also delay the completion times of many jobs. Thus, the challenge is to find the right balance between tests and executions.

For the standard version of this single-machine scheduling problem, i.e., without testing, it is well known that the Shortest Processing Time (SPT) rule is optimal for minimizing the sum of completion times. Our adversarial model is inspired by (and draws motivation from) recent work on a stochastic model of scheduling with testing introduced in [Lev16, Sha16]. They consider the problem of minimizing the weighted sum of completion times on one machine for jobs whose processing times and weights are random variables with a joint distribution, and are independent and identically distributed across jobs. In their model, testing a job does not make its processing time shorter, it only provides information for the scheduler (by revealing the exact weight and processing time for a job, whereas initially only the distribution is known). They present structural results about optimal policies and efficient optimal or near-optimal solutions based on dynamic programming.

There is a range of application settings where an operation that corresponds to a test can be applied to jobs before they are executed. For example, consider the execution of computer programs on a processor. A test could correspond to a code optimizer that takes unit time to process the program and potentially reduces its running-time. The upper limit of a job describes the running-time of the program if the code optimizer is not executed. Another application is the transmission of files over a network link. Running a compression algorithm may reduce the size of a file significantly, but the file can also be incompressible (e.g., if it is already compressed). More generally, in some systems, a job can be executed in two different modes, a *safe* mode and a *quick* mode. While the safe mode is always possible, the quick mode is not possible for every job and a test is necessary to determine it.

As a final application area, consider settings where a diagnosis can be carried out to

| competitive ratio | lower bound | upper bound | |
|---|---|---|---|
| deterministic algorithms | 1.8546 | 2 | THRESHOLD |
| randomized algorithms | 1.6257 | 1.7453 | RANDOM |
| det. alg. on uniform instances | 1.8546 | 1.9338* | BEAT |
| det. alg. on extreme uniform instances | 1.8546 | 1.8668 | UTE |
| det. alg. on extreme uniform instances with $\bar{p} \approx 1.9896$ | 1.8546 | 1.8552 | UTE |

Table 5.1: Contributions for minimizing the sum of completion times. (* holds asymptotically)

determine the exact processing time of a job. For example, fault diagnosis can determine the time needed for a repair job, or a medical diagnosis can determine the time needed for a consultation and treatment session with a patient. Assume that the person that carries out the diagnosis is the same person that executes the job and they must be allocated to a job for an uninterruptible period that is guaranteed to cover the actual time needed for the job. If the diagnosis takes unit time, we arrive at our problem of scheduling with testing.

For scheduling with testing on a single machine with the objective of minimizing the sum of completion times, we present a 2-competitive deterministic algorithm and prove that no deterministic algorithm can achieve competitive ratio less than 1.8546 in Section 5.2 We present a 1.7453-competitive randomized algorithm, showing that randomization provably helps for this problem in Section 5.3 and also give a lower bound of 1.626 on the best possible competitive ratio of any randomized algorithm. Both lower bounds hold even for instances with uniform upper limits where every processing time is either 0 or equal to the upper limit. We call such instances *extreme uniform instances*. In Section 5.4 we investigate such instances. We give a 1.8668-competitive algorithm. In the special case where the upper limit of all jobs is $\approx 1.9896$, the value used in our deterministic lower bound construction, that algorithm is even 1.8552-competitive. For the case of uniform upper limits and arbitrary processing times, we give a deterministic 1.9338-competitive algorithm. We summarize the results for this objective function in Table 5.1. Finally, we present tight results for the simpler problem of minimizing the makespan in scheduling with testing in Section 5.5. Calculations performed with the help of mathematica are provided as appendices.

## 5.1 Problem Definition and Preliminaries

The problem of scheduling with testing is defined as follows. We are given $n$ jobs to be scheduled on a single machine. Each job $j$ has an upper limit $\bar{p}_j$. It can either be executed untested (taking time $\bar{p}_j$), or be tested (taking time 1) and then executed at an arbitrary later time (taking time $p_j$, where $0 \leq p_j \leq \bar{p}_j$). Initially only $\bar{p}_j$ is known for each job, and $p_j$ is only revealed after $j$ is tested. The machine can either test or execute a job at any time. The completion time of job $j$ is denoted by $C_j$. Unless noted otherwise, we consider the objective of minimizing the sum of completion times $\sum_j C_j$.

If the processing times $p_j$ that jobs have after testing are known, an optimal schedule is easy to determine: Testing and executing job $j$ takes time $1 + p_j$, so it is beneficial to test the job only if $1 + p_j < \bar{p}_j$. In the optimal schedule, jobs are therefore ordered by non-decreasing $\min\{1 + p_j, \bar{p}_j\}$. In this order, the jobs with $1 + p_j < \bar{p}_j$ are tested and executed while jobs with $1 + p_j \geq \bar{p}_j$ are executed untested. For jobs with $1 + p_j = \bar{p}_j$ it does not matter whether the job is tested and executed, or executed untested.

As before, we use competitive analysis to evaluate our algorithms. We denote by $ALG$ the objective value (cost) of the schedule produced by an algorithm and by $OPT$ the optimal cost. An algorithm is $\rho$-competitive or has competitive ratio at most $\rho$, if $ALG/OPT \leq \rho$ for all instances of the problem. For randomized algorithms, $ALG$ is replaced by $E[ALG]$ in this definition.

When we analyze an algorithm or the optimal schedule, we will typically first argue that the schedule has a certain structure with different blocks of tests or job completions. Once we have established that structure, the cost of the schedule can be calculated by adding the cost for each block taken in isolation, plus the effect of the block on the completion times of later jobs. For example, assume that we have $n$ jobs with upper limit $\bar{p}$, that $\alpha n$ of these jobs are *short*, with processing time 0, and $(1-\alpha)n$ jobs are *long*, with processing time $\bar{p}$. If an algorithm (in the worst case) first tests the $(1-\alpha)n$ long jobs, then tests the $\alpha n$ short jobs and executes them immediately, and finally executes the $(1-\alpha)n$ long jobs that were tested earlier (see also Figure 5.1), the total cost of the schedule can be calculated as

$$(1-\alpha)n^2 + \frac{\alpha n(\alpha n+1)}{2} + \alpha n(1-\alpha)n + \frac{(1-\alpha)n((1-\alpha)n+1)}{2}\bar{p},$$

where $(1-\alpha)n^2$ is the total delay that the $(1-\alpha)n$ tests of long jobs add to the completion times of all $n$ jobs, $\frac{\alpha n(\alpha n+1)}{2}$ is the sum of completion times of a block with $\alpha n$ short jobs

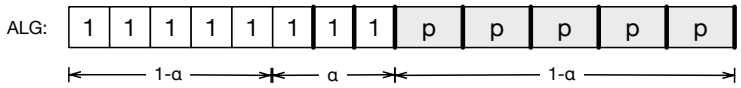



Figure 5.1: Typical schedule produced by an algorithm. White jobs are tests and grey jobs are actual jobs. The completion time of a job is depicted by a thick bar. Test and execution of a job might be separated. A job of length 0 completes immediately after its test.

that are tested and executed, $\alpha n(1-\alpha)n$ is the total delay that the block of short jobs with total length $\alpha n$ adds to the completion times of the $(1-\alpha)n$ jobs that come after it, and $\frac{(1-\alpha)n((1-\alpha)n+1)}{2}\bar{p}$ is the sum of completion times for a block with $(1-\alpha)n$ job executions with processing time $\bar{p}$ per job.

**Lower limits.** A natural generalization of the problem would be to allow each job $j$ to have, in addition to its upper limit $\bar{p}_j$, also a lower limit $\ell_j$, such that the processing time after testing satisfies $\ell_j \leq p_j \leq \bar{p}_j$. We observe that the presence of lower limits has no effect on the optimal schedule, and can only help an algorithm. As we are interested in worst-case analysis, we assume in the following that every job has a lower limit of 0. Any algorithm that is $\rho$-competitive in this case is also $\rho$-competitive in the case with arbitrary lower limits (the algorithm can simply ignore the lower limits).

**Jobs with small $\bar{p}_j$.** We will consider several algorithms and prove competitiveness for them. We observe that any $\rho$-competitive algorithm may process jobs with $\bar{p}_j < \rho$ without testing in order of increasing $\bar{p}_j$ at the beginning of its schedule.

Lemma 5.1 *Without loss of generality any algorithm ALG (deterministic or randomized) claiming competitive ratio $\rho$ starts by scheduling untested all jobs $j$ with $\bar{p}_j < \rho$ in increasing order of $\bar{p}_j$. Also worst-case instances for ALG consist solely of jobs $j$ with $\bar{p}_j \geq \rho$.*

*Proof.* We transform *ALG* into an algorithm *ALG′* which obeys the claimed behavior and show that its ratio does not exceed $\rho$. Consider an arbitrary instance $I$. Let $J$ be the sequence of jobs $j$ with $\bar{p}_j < \rho$ ordered by increasing $\bar{p}_j$. We divide the sequence $J$ into $J_0J_1$, where $J_0$ consists of the jobs $j$ with $0 \leq \bar{p}_j < 1$ and $J_1$ consists of the jobs $j$ with $1 \leq \bar{p}_j < \rho$. *ALG′* starts by executing the job sequence $J$ untested. In a worst-case instance all these jobs have processing time 0. By optimality of the SPT policy *OPT*

also schedules $J_0$ first and untested, and then schedules $J_1$ tested spending time 1 on each job. The ratio of the costs of these parts is

$$\frac{ALG'(J)}{OPT(J)} < \rho,$$

where the inequality follows from $\bar{p}_j / \min\{1, \bar{p}_j\} < \rho$ for all $j \in J$. Let len denote the length of a schedule. Then by the same argument we have

$$\frac{\text{len}(ALG'(J))}{\text{len}(OPT(J))} < \rho.$$

Let $I'$ be the instance $I$ without the jobs in $J$ and let $k$ be the number of jobs in $I'$. We assume $k > 0$ otherwise we are done with the proof. Since $I'$ contains only jobs with upper limit at least $\rho$, we have $ALG(I') = ALG'(I')$. Then holds

$$ALG'(I) = ALG'(J) + k \cdot \text{len}(ALG'(J)) + ALG'(I')$$
$$OPT(I) = OPT(J) + k \cdot \text{len}(OPT(J)) + OPT(I').$$

From these (in)equalities we conclude

$$\frac{ALG(I)}{OPT(I)} \leq \rho \Rightarrow \frac{ALG'(I)}{OPT(I)} \leq \rho$$
$$\frac{ALG(I)}{OPT(I)} \geq \rho \Rightarrow \frac{ALG'(I)}{OPT(I)} \leq \frac{ALG(I')}{OPT(I')}.$$

This means if $ALG$ is $\rho$ competitive then so is $ALG'$ and that there are worst-case instances for $ALG$ only with jobs having upper limit at least $\rho$. □

**Increasing or decreasing ALG and OPT.** We sometimes consider worst-case instances consisting of only a few different job types. The following proposition allows us to do so in some cases.

Proposition 5.2 *Fix some algorithm ALG and consider a family of instances described by some parameter $x \in [\ell, u]$, which could represent $p_j$ or $\bar{p}_j$ for some job $j$ or for some set of jobs. Suppose that both OPT and ALG are linear in $x$ for the range $[\ell, u]$. Then the ratio $ALG/OPT$ does not decrease for at least one of the two choices $x = \ell$ or $x = u$. Moreover, if OPT and ALG are increasing in $x$ with the same slope, then this holds for $x = \ell$.*

*Proof.* The proof follows from the fact that an expression of the form $ALG/OPT = (a + bx)/(a' + b'x)$ is monotone in $x$. Indeed its derivative is

$$\frac{a'b - ab'}{(a' + b'x)^2}$$

whose sign does not depend on $x$. The last statement follows from the fact that if $ALG > OPT$ and $0 < \delta \leq OPT$, then $(ALG - \delta)/(OPT - \delta) > ALG/OPT$. □

We can make successive use of this proposition to show useful properties on worst-case instances.

Lemma 5.3 *Suppose that there is an interval $[\ell', u']$ such that OPT schedules all jobs $j$ with $p_j \in [\ell', u']$ either all tested or all untested and this independent of the actual processing time in $[\ell', u']$. Suppose this also holds for ALG. Moreover suppose that both OPT and ALG are insensitive to changes of the processing times in $[\ell', u']$ which maintain the ordering of processing times. Then there is a worst-case instance for ALG where every job $j$ with $p_j \in [\ell', u']$ satisfies $p_j \in \{\ell', u'\}$.*

*Proof.* Fix some worst-case instance for the algorithm $ALG$. Let $S$ be the set of jobs $j$ with $p_j = x$. Let $\ell$ be the largest processing time strictly smaller than $x$ or $\ell'$ if $x$ is already the smallest processing time or if this would make $\ell$ smaller than $\ell'$. Also let $u$ be the largest processing time strictly larger than $x$ or $u'$ if $x$ is already the largest processing time or if this would exceed $u'$. Formally $\ell = \max(\{\ell'\} \cup \{p_i : p_i < x\})$ and $u = \min(\{u'\} \cup \{p_i : p_i > x\})$. Since the schedules are preserved when changing the processing times of $S$, both costs $ALG$ and $OPT$ are linear in $x$ within $[\ell, u]$. Now we can use Proposition 5.2 to show that there is a worst-case instance where all jobs in $S$ have processing time either $\ell$ or $u$. In both cases we have reduced the number of distinct processing times strictly being between $\ell'$ and $u'$. By repeating this argument sufficiently often we obtain the claimed statement. □

## 5.2 Deterministic Algorithms

Scheduling jobs on one machine to minimize the sum of completion times can be solved optimally using the Shortest Processing Time (SPT) rule. In our adversarial model with testing, the optimal schedule follows this principle and tests all jobs for which the test reduces the processing time by at least 1. Any algorithm has to decide online which jobs to test and in which order to execute the jobs. At each time point it only knows the upper limit of all jobs and the processing time of the jobs that have been tested. We show a natural algorithm that achieves competitive ratio 2. Then we prove that no deterministic algorithm has competitive ratio less than 1.8546.

### 5.2.1 Algorithm Threshold

We show a competitive ratio of 2 for a natural algorithm that uses a threshold to decide whether to test a job or execute it untested.

**Algorithm (Threshold)** First, jobs with $\bar{p}_j < 2$ are scheduled in order of non-decreasing upper limits without test. Then all remaining jobs are tested. If the revealed processing time of job $j$ is $p_j \leq 2$ (short jobs), then the job is executed immediately after its test. After all remaining jobs have been tested, the pending jobs (long jobs) are scheduled in order of increasing processing time $p_j$.

By Lemma 5.1 we may restrict our competitive analysis w.l.o.g. to instances with $\bar{p}_j \geq 2$. Note, that on such instances Threshold tests all jobs. From a simple interchange argument it follows that the structure of the algorithm's solution in a worst-case instance is as follows:

- Test phase: The algorithm tests all jobs that have $p_j > 2$, and defers them.
- Short jobs phase: The algorithm tests short jobs ($p_j \leq 2$) and executes each of them right away. The jobs are tested in order of non-increasing processing time.
- Long jobs phase: The algorithm executes all deferred long jobs in order of non-decreasing processing times.

An optimal solution will not test jobs with $p_j + 1 \geq \bar{p}_j$. It sorts jobs in non-decreasing order of values $\min\{1 + p_j, \bar{p}_j\}$.

First, we analyze and simplify worst-case instances.

**Lemma 5.4** *There is a worst-case instance for* Threshold *in which all short jobs with $p_j \leq 2$ have processing time either 0 or 2.*

We give a proof without modifying upper limits, which is not necessary in this section but will come handy later when we analyze Threshold for arbitrary uniform upper limits.

*Proof.* Consider short jobs that are tested by both, the optimum and Threshold, i.e., short jobs with $p_j < \bar{p}_j - 1$. We argue that we can either decrease the processing time of a short job $j$ to 0 or increase it to $\min\{2, \bar{p}_j - 1\}$ without decreasing the worst-case ratio. Consider Threshold and let $\ell$ be the first short job with $p_\ell < \min\{2, \bar{p}_\ell - 1\}$ and let $i$ be the last short job with $p_i > 0$.

Suppose $i \neq \ell$. Let $\Delta = \min\{p_i, \min\{2, \bar{p}_\ell - 1\} - p_\ell\}$. We decrease $p_i$ by $\Delta$ and at the same time increase $p_\ell$ by $\Delta$. The value $\Delta$ is chosen in such a way that either $p_i$ will become 0 or $p_\ell$ will be $\min\{2, \bar{p}_\ell - 1\}$, as desired. The schedule produced by the algorithm will be the same except that jobs $\ell, \ldots, i-1$ complete $\Delta$ units later. In the optimal schedule $\ell$ and $i$ are scheduled in opposite order. Suppose we keep the schedule fixed when changing the processing times of jobs $i$ and $\ell$. Then $i$'s completion time as well as those of jobs between $i$ and $\ell$ decreases. In an optimal schedule jobs might be re-ordered, but this only improves the total objective further. Hence, the total ratio of objective values does not decrease.

Now, assume $i = \ell$, i.e., there is exactly one short job with processing time $p_i$ strictly between 0 and $\min\{2, \bar{p}_i - 1\}$. We argue that either increasing or decreasing $p_i$ to $\min\{2, \bar{p}_i - 1\}$ or 0 will not decrease the worst-case ratio. Such a change $\Delta$ does not change the order of jobs in the algorithm's solution and thus the change in the objective is $\Delta$ times the number of jobs completing after $i$. In an optimal solution, there are untested short or long jobs which are scheduled between short tested jobs and their relative order with $i$ may change when $i$ is in-/decreased by $\Delta$. However, let us consider a possibly not optimal schedule that simply does not adjust the order after changing $i$. Then the change in the objective is linear in $\Delta$ in the above-given range, as it is for the algorithm, and thus, by Proposition 5.2, either increasing or decreasing $p_i$ by $\Delta$ does not decrease the ratio of objective values. Now, the truly optimal objective value is not larger and thus, the true worst-case ratio is not smaller.

Now, we may assume that all short jobs remaining with processing times different from 0 and 2 are untested in the optimum solution because their processing time is at least $\bar{p}_j - 1$. Again, the optimum does not test those jobs, and hence, increasing the processing time to 2 has no impact on the optimal schedule, while our algorithm's cost only increases. Thus, the worst-case ratio increases. □

Threshold tests all jobs and takes scheduling decisions depending on job processing times $p_j$ but independently of upper limits of jobs. Since all short jobs have $p_j \in \{0, 2\}$, we can reduce all their upper limits to $\bar{p}_j = 2$ without affecting the algorithm schedule, whereas it may only improve the optimal schedule. In particular we may assume now the following.

**Proposition 5.5** *There is a worst-case instance in which all short jobs have $\bar{p}_j = 2$ and execution times are 0 or 2.*

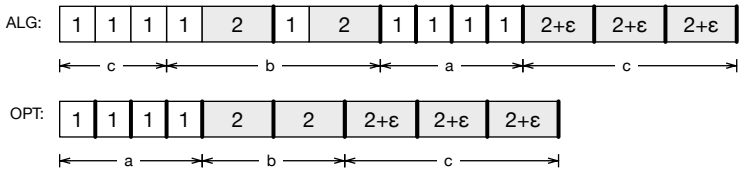



Figure 5.2: Worst case instance for Threshold.

**Lemma 5.6** *There is a worst-case instance in which long jobs with $p_j > 2$ satisfy $p_j = \bar{p}_j = 2 + \varepsilon$ for infinitesimally small $\varepsilon > 0$.*

*Proof.* For all long jobs, which are tested in the optimal schedule, we reduce the upper limit to $\bar{p}_j = 1 + p_j$. This does not change the algorithm's solution, but the optimum may as well run those previously tested jobs also untested and without changing its total objective value. Now the optimum solution runs all long jobs without testing them. Thus, increasing the processing time of long jobs to $p_j = \bar{p}_j$ does not affect the optimum cost whereas the algorithm's cost increase.

Proposition 5.5 implies that all long jobs are scheduled in the same order by the algorithm and an optimum without any small jobs in between. Then, setting $\bar{p} = 2 + \varepsilon$ decreases the objective values of both algorithms by the same amount and thus does not decrease the ratio. □

Now we are ready to prove the main result.

**Theorem 5.7** *Algorithm* Threshold *has competitive ratio at most* 2.

*Proof.* We consider worst-case instances of the type derived above. Let $a$ be the number of short jobs with $p_j = 0$, let $b$ be the number of short jobs with $\bar{p}_j = p_j = 2$, and let $c$ be the number of long jobs with $\bar{p}_j = 2 + \varepsilon$, see Figure 5.2.

Threshold's solution for a worst-case instance first tests all long jobs, then tests and executes the short jobs in decreasing order of processing times, and completes with the executions of long jobs. The total objective value $ALG$ is

$$ALG = (a + b + c)c + b(b + 1)/2 \cdot 3 + 3b(a + c) + a(a + 1)/2 + a \cdot c + c(c + 1)/2 \cdot (2 + \varepsilon).$$

An optimum solution tests and schedules first all 0-length jobs and then executes the remaining jobs without tests. The objective value is

$$OPT = a(a + 1)/2 + a(b + c) + b(b + 1)/2 \cdot 2 + 2bc + c(c + 1)/2 \cdot (2 + \varepsilon).$$

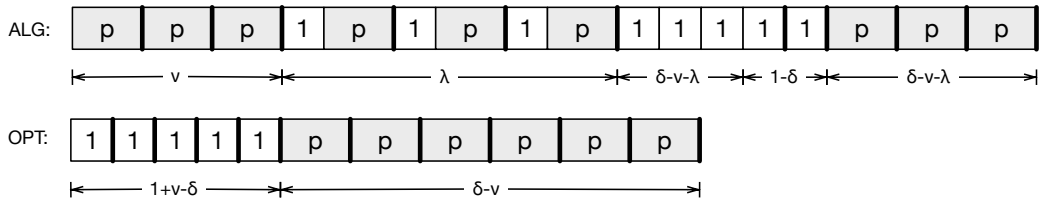



Figure 5.3: Lower bound construction

Simple transformation shows that $ALG \leq 2 \cdot OPT$ is equivalent to

$$2ab + 2c^2 \leq a^2 + b^2 + a + b + c(c+1)(2+\varepsilon) \;\Leftrightarrow\; 0 \leq (a-b)^2 + a + b + c^2\varepsilon + c(2+\varepsilon),$$

which is obviously satisfied and the theorem follows. □

### 5.2.2 Deterministic Lower Bound

In this section we give a lower bound on the competitive ratio of any deterministic algorithm. The instances constructed by the adversary have a very special form: All jobs have the same upper limit $\bar{p}$, and the processing time of every job is either 0 or $\bar{p}$.

Consider instances of $n$ jobs with uniform upper limit $\bar{p} > 1$, and consider any deterministic algorithm. We say that the algorithm *touches* a job when it either tests the job or executes it untested. We re-index jobs in the order in which they are first touched by the algorithm, i.e., job 1 is the first job touched by the algorithm and job $n$ is the last. The adversary fixes a fraction $\delta \in [0, 1]$ and sets the processing time of job $j$, $1 \leq j \leq n$, to:

$$p_j = \begin{cases} 0 & \text{, if } j \text{ is executed by the algorithm untested, or } j > \delta n \\ \bar{p} & \text{, if } j \text{ is tested by the algorithm and } j \leq \delta n \end{cases}.$$

We call a job $j$ is called *short* if $p_j = 0$ and *long* if $p_j = \bar{p}$.

We assume the algorithm knows $\bar{p}$ and $\delta$, which can only improve the performance of the best-possible deterministic algorithm. Note that with $\delta$ and $\bar{p}$ known to the algorithm, it has full information about the actions of the adversary. Nevertheless, it is still non-trivial for an algorithm to decide for each of the first $\delta n$ jobs whether to test it (which makes the job a long job, and hence the algorithm spends time $\bar{p} + 1$ on it while the optimum executes it untested and spends only time $\bar{p}$) or to execute it untested (which makes it a short job, and hence the algorithm spends time $\bar{p}$ on it while the optimum spends only time 1).

Let us first determine the structure of the schedule produced by an algorithm that achieves the best-possible competitive ratio for instances created by this adversary, as displayed in Figure 5.3.

Lemma 5.8 *The schedule of a deterministic algorithm with best possible competitive ratio has the following form, where $\lambda, \nu \geq 0$ and $\nu + \lambda \leq \delta$: The algorithm first executes $\nu n$ jobs untested, then tests and executes $\lambda n$ long jobs, then tests $(\delta - \nu - \lambda)n$ long jobs and delays their execution, then tests and executes the remaining $(1 - \delta)n$ short jobs, and finally executes the $(\delta - \nu - \lambda)n$ delayed long jobs that were tested earlier.*

*Proof.* It is clear that the algorithm will test the last $(1 - \delta)n$ jobs and execute each such job (with processing time 0) right after its test, as executing any of them untested does not affect the optimal solution but increases the objective value of the algorithm. Furthermore, consider the time $t$ when the algorithm tests job $j_0$. From this time until the end of the schedule, the algorithm will test and execute the last $(1 - \delta)n$ jobs (spending time 1 on each such job), and execute all the long jobs that were tested earlier but not yet executed (spending time $\bar{p} > 1$ on each such job). As the SPT rule is optimal for minimizing the sum of completion times, it is clear that from time $t$ onward the algorithm will first test and execute the $(1 - \delta)n$ short jobs and afterwards execute the long jobs that were tested but not executed before time $t$.

Before time $t$, the algorithm touches the first $\delta n$ jobs. Each of these can be executed untested (let $\nu n$ be the number of such jobs), or tested and also executed before time $t$ (let $\lambda n$ be the number of such jobs), or tested but not executed before time $t$ (this happens for the remaining $(\delta - \nu - \lambda)n$ jobs). To minimize the sum of completion times of these jobs, it is clear that the algorithm first executes the $\nu n$ jobs untested (spending time $\bar{p}$ per job), then tests the $\lambda n$ long jobs and executes each of them right after its test (spending time $1 + \bar{p}$ per job), and finally tests the remaining $(\delta - \nu - \lambda)n$ long jobs. □

The cost of the algorithm in dependence on $\nu$, $\lambda$, $\delta$ and $\bar{p}$ can now be expressed as:

$$ALG(\nu, \lambda, \delta, \bar{p}) = n^2/2 \cdot \Big[\bar{p}\nu^2 + 2\bar{p}\nu(1 - \nu) + (1 + \bar{p})\lambda^2 + 2(1 + \bar{p})\lambda(1 - \nu - \lambda) + 2(\delta - \nu - \lambda)(1 - \nu - \lambda) + (1 - \delta)^2 + 2(1 - \delta)(\delta - \nu - \lambda) + \bar{p}(\delta - \nu - \lambda)^2\Big] + O(n).$$

The optimal schedule first tests and executes the $(\nu + 1 - \delta)n$ short jobs and then executes the $(\delta - \nu)n$ long jobs untested. Hence, the optimal cost, which depends only on $\nu$, $\delta$ and $\bar{p}$, is:

$$OPT(\nu, \delta, \bar{p}) = n^2/2 \cdot \Big((\nu + 1 - \delta)^2 + 2(\nu + 1 - \delta)(\delta - \nu) + \bar{p}(\delta - \nu)^2\Big) + O(n).$$

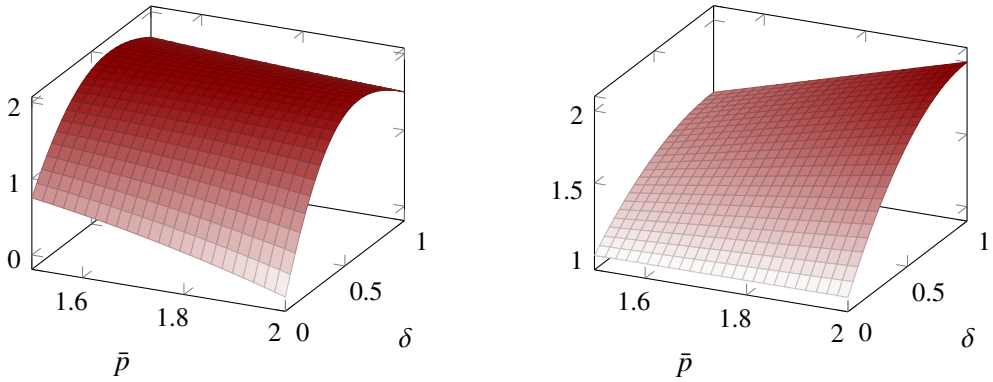



Figure 5.4: The competitive ratio $\rho_1$ (left) and $\rho_2$ (right).

As the adversary can choose $\delta$ and $\bar{p}$, while the algorithm chooses $\nu$ and $\lambda$, the value

$$\rho = \max_{\delta,\bar{p}} \min_{\nu,\lambda} \lim_{n\to\infty} \frac{ALG(\nu,\lambda,\delta,\bar{p})}{OPT(\nu,\delta,\bar{p})}$$

gives a lower bound on the competitive ratio of any deterministic algorithm in the limit for $n \to \infty$. By making $n$ sufficiently large, the adversary can create instances with finite $n$ that give a lower bound that is arbitrarily close to $\rho$.

Theorem 5.9 *No deterministic algorithm can achieve a competitive ratio below* 1.8546. *This holds even for instances with uniform upper limit where each processing time is either* 0 *or equal to the upper limit.*

*Proof.* We use the help of mathematica to optimize the choice of $\delta$ and $\bar{p}$. First, we compute the best response of the algorithm for any fixed $\delta$ and $\bar{p}$ and find there are two local minima. If the algorithm chooses $\lambda = 0$ and $\nu = \delta$, we have

$$\rho_1 = \frac{-\bar{p}^2(1-\delta)^2 + (2-\delta)\delta + \bar{p}(2-\delta(2-\delta))}{1+(\bar{p}-1)\delta^2}.$$

For $\delta < 1 - 1/\bar{p}$, this is the only local minimum and it is at most 1.75 for any adversarial choice of $\delta \in [0, 1-1/\bar{p})$ and $\bar{p} \in [1.5, 2]$. For $\delta \geq 1 - 1/\bar{p}$, a second local minimum exists where the first derivative of the ratio in $\lambda$ is 0 and $\nu = 0$. In this case the competitive ratio is

$$\rho_2 = 1 + (\bar{p}-1)(2-\delta)\delta.$$

We display both functions in Figure 5.4. The best-response algorithm has the minimum of these two functions as its competitive ratio. To maximize the competitive ratio, we

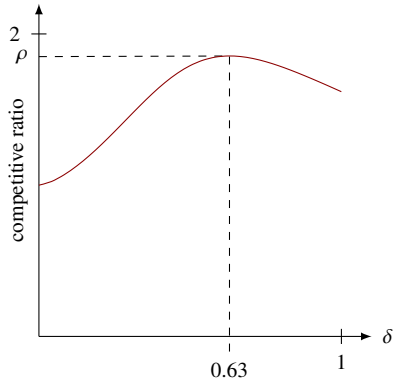



Figure 5.5: Competitive ratio depending on $\delta$

choose $\bar{p}$ depending on $\delta$, such that $\rho_1 = \rho_2$. Then the competitive ratio only depends on $\delta$ and equals

$$\rho = \frac{-2 - (2-\delta)\delta(-2+\delta^2 + \sqrt{\delta}\sqrt{8 + (-2+\delta)\delta(10 + \delta(-3 + 4(-2+\delta)\delta))})}{2(-1 + (-2+\delta)(-1+\delta)\delta(1+\delta))}.$$

It attains its maximum when $\delta$ is the only real root in the interval $[0, 1]$ of the polynomial $-18 + 93\delta - 212\delta^2 + 277\delta^3 - 197\delta^4 + 26\delta^5 + 82\delta^6 - 75\delta^7 + 20\delta^8 + 8\delta^9 - 6\delta^{10} + \delta^{11}$. This is approximately 0.6307 and yields a lower bound of roughly 1.8546 as displayed in Figure 5.5. □

## 5.3 Randomized Algorithms

Randomization is a useful tool in adversarial models. It weakens an advantage the adversary has over an algorithm, which some consider unfair. Namely, given several jobs with equal upper limit, they are indistinguishable for any algorithm and thus occur in their worst-possible order in an adversarial sequence. For the optimal solution they are distinguishable by the processing time they reveal after a test. Randomization now allows an algorithm to choose one of the indistinguishable jobs at random, which improves upon the worst-case ordering. Additionally, randomized algorithms may also take any other decision such as testing, executing, or delaying a job at random.

We describe an algorithm that uses only the first of these two possibilities and show in expectation it performs better than any performance guarantee a deterministic algorithm can achieve. Using Yao's principle we also prove a lower bound on the competitive ratio of randomized algorithms.

### 5.3.1 Algorithm Random

We refine the ideas we used for the deterministic algorithm Threshold to design a randomized algorithm. As before we define a threshold $T$ to decide whether to test a job or to execute it untested. The order in which the jobs are tested is random. We use a second threshold $E$ to distinguish jobs that we execute immediately after their test and those whose execution we delay. While for Threshold the two thresholds are the same, here they will turn out to be different.

**Algorithm (Random)** The randomized algorithm Random has parameters $1 \leq T \leq E$ and works in three phases. First, it executes all jobs with $\bar{p}_j < T$ without test in order of increasing $\bar{p}_j$. Then it tests all jobs with $\bar{p}_j \geq T$ in uniform random order. Each tested job $j$ is executed immediately after its test if $p_j \leq E$ and is deferred otherwise. Finally, all deferred jobs are executed in order of increasing processing time.

We analyze the competitive ratio of Random, and optimize the parameters $T, E$ such that the resulting competitive ratio is $T$. By Lemma 5.1 we restrict to instances with $\bar{p}_j \geq T$ for all jobs. Then, the schedule produced by Random can be divided into two parts. Part (1) contains all tests, of which those that yield processing time $p_j$ at most $E$ are immediately followed by the job's execution. Part (2) contains all jobs that *have been* tested and have processing time larger than $E$. These jobs are ordered by increasing processing time. Jobs in the first part are completed in an arbitrary order.

**Lemma 5.10** *There is a worst-case instance for* Random *containing only the following four jobs types: jobs with upper limit $\bar{p}_j = T$ and processing time $p_j \in \{0, T\}$, jobs with upper limit and processing time $\bar{p}_j = p_j = E$ and those with upper limit and processing time $\bar{p}_j = p_j = E + \varepsilon$.*

*Proof.* We take an arbitrary instance and modify the upper limits and processing times until only these four job types remain. First observe that we can assume $\bar{p}_j = \max\{p_j, T\}$ for all jobs. Reducing $\bar{p}_j$ to this value does not change the cost or behavior of Random, but may decrease the cost of $OPT$. Let $\epsilon > 0$ be an arbitrary small number such that $p_j \geq E + \epsilon$ for all jobs $j$ with $p_j > E$. These jobs are executed by Random in part (2) of the schedule in non-decreasing order of processing time. The same holds for $OPT$, which by the *SPT Rule* also schedules these jobs at the end in exactly the same order. Hence, if we set $\bar{p}_j = p_j = E + \epsilon$ for all these jobs, then we reduce the objective value of

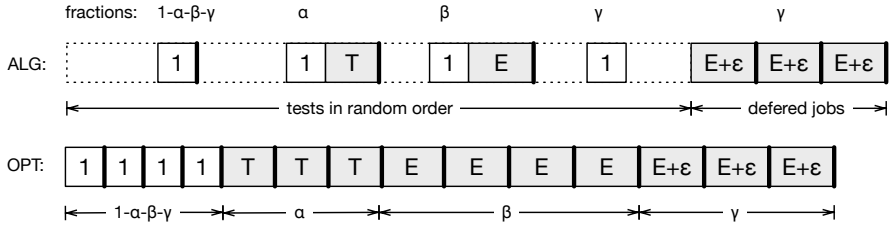



Figure 5.6: Worst case analysis of the algorithm RANDOM.

RANDOM and of $OPT$ by the same amount. By Proposition 5.2 this transformation only increases the competitive ratio of the algorithm.

We apply Lemma 5.3 to show that for all jobs $j$ with $p_j \in [T, E]$ we can in fact assume $p_j \in \{T, E\}$. The use of the lemma is a bit subtle as the output of RANDOM is a distribution of schedules. For each fixed order the conditions of the statement of the lemma are satisfied. As the expected completion time of RANDOM is a linear combination of the objective values over each of the $n!$ orders, the lemma holds.

Now we turn to jobs $j$ with $\bar{p}_j = T$ and $p_j \leq T$. For the jobs with $0 \leq p_j \leq T - 1$, the same argument implies that $p_j \in \{0, T - 1\}$. Jobs $j$ with $\bar{p}_j = T$ and $T - 1 \leq p_j \leq T$ are not tested in $OPT$. Therefore, increasing their processing time to $p_j = T$ does not change $OPT$ but increases the cost of RANDOM and consequently increases the competitive ratio. □

In conclusion, a worst case instance is described completely by the number of jobs $n$ and fractions $\alpha, \beta, \gamma \in [0, 1]$ as follows, see Figure 5.6.

- A $1 - \alpha - \beta - \gamma$ fraction of the jobs have $\bar{p}_j = T$ and $p_j = 0$. (type 0 jobs)
- An $\alpha$ fraction of the jobs have $\bar{p}_j = p_j = T$. (type T jobs)
- A $\beta$ fraction of the jobs have $\bar{p}_j = p_j = E$. (type E jobs)
- A $\gamma$ fraction of the jobs have $\bar{p}_j = p_j = E + \epsilon$ for some arbitrarily small $\epsilon > 0$. (type $E^+$ jobs)

We first consider the expected algorithm cost and omit $\epsilon$ for simplicity. We denote by $L := n + T\alpha n + E\beta n$ the length of part (1). This means that for a job $j$ of type $0, T$ or $E$, the expected time its test starts is $(L - 1 - p_j)/2$ and hence its expected completion time, which is $1 + p_j$ time units later, is $(L + 1 + p_j)/2$. Jobs completed in the second part have all the same processing time. The $i$-th job that is completed in part (2) has completion

time $L + Ei$. Thus, the expected objective value of RANDOM can be expressed as

$$E[ALG] = n^2/2 \cdot \left[(1-\gamma)(1+T\alpha+E\beta)^2 + 2\gamma(1+T\alpha+E\beta) + E\gamma^2\right] + n/2 \cdot [1-\gamma+T\alpha+E\beta+E\gamma)].$$

By the *SPT rule*, the optimal schedule first tests and executes all jobs of type 0. Then it executes the jobs of type $T, E$ and $E^+$ jobs without test in that order. Hence the optimal objective value is stated as follows, where every other expression represents the total completion times of some job type followed by the delay these jobs induce on subsequent job types and the linear terms in $n$ follow at the end

$$OPT = n^2/2 \cdot \left[(1-\alpha-\beta-\gamma)^2 + 2(1-\alpha-\beta-\gamma)(\alpha+\beta+\gamma) + T\alpha^2 + 2T\alpha(\beta+\gamma) + E\beta^2 + 2E\beta\gamma + E\gamma^2\right] + n/2 \cdot \left[(1-\alpha-\beta-\gamma) + T\alpha + E\beta + E\gamma\right].$$

**Theorem 5.11** *The algorithm* RANDOM *has competitive ratio at most* 1.7453*, if we choose* $T \approx 1.7453$ *and* $E \approx 2.8609$*. The exact, best-possible, parameter choices are* $T$ *equals the root of the polynomial* $-1 - 16T + 20T^2 + 36T^3 - 52T^4 + 16T^5$ *in* $[1.5, 2]$ *and*

$$E = \frac{T(3 - 8T + 4T^2)}{T - 1}.$$

*Proof.* We say that fractions $\alpha, \beta, \gamma$ are *valid* if and only if $\alpha, \beta, \gamma \geq 0$ and $\alpha + \beta + \gamma \leq 1$. The algorithm is $T$-competitive, if $T \cdot OPT - E[ALG] \geq 0$ for all $n \geq 0$ and all valid fractions $\alpha, \beta, \gamma$. The costs can be written as $E[ALG] = \frac{n^2}{2}E[ALG_2] + \frac{n}{2}E[ALG_1]$ and $OPT = \frac{n^2}{2}OPT_2 + \frac{n}{2}OPT_1$ for

$$\begin{aligned} E[ALG_2] &= 1 + \gamma + E\left(\beta + \beta\gamma + \gamma^2\right) + T(\alpha + \alpha\gamma) \\ E[ALG_1] &= 1 - \gamma + \beta E + \gamma E + \alpha T \\ OPT_2 &= 1 + (\beta+\gamma)^2(E-1) + \alpha^2(T-1) + 2\alpha(\beta+\gamma)(T-1) \\ OPT_1 &= 1 - \alpha - \beta - \gamma + \beta E + \gamma E + \alpha T. \end{aligned}$$

The inequality $T \cdot OPT_2 - ALG_2 \geq 0$ is a necessary condition, as it describes the competitive ratio for $n \to \infty$. We use it to find the optimal choice for $T$ and $E$. Then we prove for these values also $T \cdot OPT_1 - ALG_1 \geq 0$ holds for all valid $\alpha, \beta, \gamma$ fractions, which means RANDOM is $T$-competitive. We denote the first inequality by $G$:

$$G = T \cdot \left[1 + (\beta+\gamma)^2(E-1) + \alpha^2(T-1) + 2\alpha(\beta+\gamma)(T-1) - \alpha - \alpha\gamma\right] - \gamma - 1 - E\left(\gamma^2 + \beta\gamma + \beta\right).$$

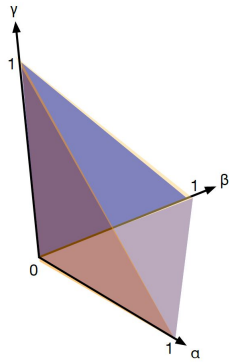



Figure 5.7: Validity region for $(\alpha, \beta, \gamma)$.

We want to find parameters $T, E$ with minimal $T$ such that $G(T, E, \alpha, \beta, \gamma) \geq 0$ for all valid fractions, i.e. $\alpha, \beta, \gamma \geq 0$ with $\alpha + \beta + \gamma \leq 1$. We call this the *validity polytope* for $\alpha, \beta, \gamma$, see Figure 5.7. For this purpose we made numerical experiments which gave us a range where the optima could belong, namely $T \in [1.71, 1.78], E \in [2.82, 2.89]$.

Our general approach consists in identifying values $(\alpha, \beta, \gamma)$ which are local minima for $G$. Each of these points $(\alpha, \beta, \gamma)$ generates a condition on $T, E$ of the form $G(T, E, \alpha, \beta, \gamma) \geq 0$. The optimal pair $(T, E)$ is then the pair with minimal $T$ satisfying all the generated conditions.

We prove in the following lemma, that Figure 5.8 depicts the conditions that need to be fulfilled. The Conditions (5.2) and (5.4) are most restrictive. We compute their left-most crossing point and find the best choices are $T$ equals the root in $[1.5, 2]$ of the polynomial $-1-16T+20T^2+36T^3-52T^4+16T^5$ and $E$ is $T(3-8T+4T^2)/(T-1)$. This is approximately $T \approx 1.7453$ and $E \approx 2.8609$. We conclude the proof by considering the inequality $T \cdot OPT_1 - ALG_1 \geq 0$ which is

$$\gamma(E-1)(T-1) + \beta(E-1)t + (1-\alpha(2-T))T - 1 - \beta E \geq 0.$$

Taking the derivative of the left hand side reveals that it is decreasing in $\alpha$ and increasing in $\beta$ and $\gamma$ for the chosen values $T, E$. Hence the expression is minimized at $\alpha = 1, \beta = 0, \gamma = 0$, where its value is $T(T-1) - 1 > 0$. Therefore, given Lemma 5.12, we have proven the theorem. □

Lemma 5.12 *$T \approx 1.7453$ and $E \approx 2.8609$ is the parameter pair with smallest $T$, for which $G$ is non-negative everywhere in the validity polytope.*

*Proof.* We partition the validity polytope. First; we consider the open region $\{(\alpha, \beta, \gamma) | 0 < \alpha, 0 < \beta, 0 < \gamma, \alpha + \beta + \gamma < 1\}$. Then; we consider the four open facets on the border

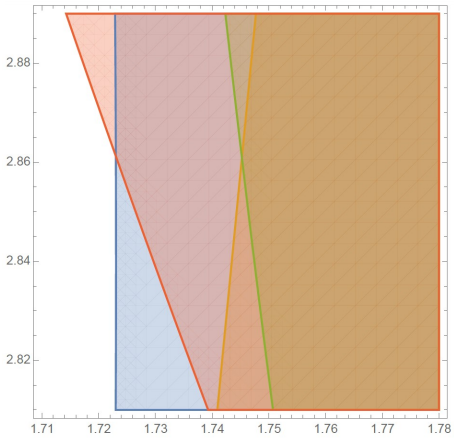



Figure 5.8: Regions where conditions (5.1):blue, (5.2):orange, (5.4):green and (5.8):red are satisfied by points $(T, E)$, with $T$ ranging horizontally and $E$ ranging vertically.

defined by the equations $\alpha+\beta+\gamma=1, \alpha=0, \beta=0, \gamma=0$. Finally, we consider the 6 closed edges that form the edges of the polytope. Note that the vertices of the polytope $(0,0,0),(0,0,1),(0,1,0),(1,0,0)$ belong each to several edges.

**open polytope.** The second order derivatives of $G$ in $\alpha, \beta, \gamma$ are

$$\frac{\partial^2 G}{\partial^2 \alpha} = 2T(T-1) \qquad \frac{\partial^2 G}{\partial^2 \beta} = 2T(E-1) \qquad \frac{\partial^2 G}{\partial^2 \gamma} = 2T(E-1) - 2E,$$

which are all positive in the considered $T$- and $E$-range. Hence, a local minimum on the open polytope must be a point $(\alpha, \beta, \gamma)$ that is a root for the derivative in each of the three directions. Hence, we choose

$$\alpha = \beta - \gamma + \frac{1+\gamma}{2(T-1)} \qquad \beta = \frac{1+\gamma-2\gamma T}{2T} \qquad \gamma = \frac{(E(T-1)-T)(2T-1)}{E(T-1)+T}.$$

For this point the condition $G \geq 0$ translates into the following condition on $T, E$:

$$E^2(T-1)^2 + T(2T-1) - ET^2 \geq 0. \tag{5.1}$$

**open facet $\alpha+\beta+\gamma=1$.** In that case the derivative of $G$ in $\beta$ is $1-\alpha(E-T)$. This means that $G$ is linear in $\beta$, and no local minimum lies inside of the triangle. Note that in the degenerate case $\alpha = 1/(E-T)$ the value of $G$ is independent of $\beta$, hence it is enough to consider an equivalent point on the boundary, which will be considered below.

**open facet $\gamma=0$.** In this case the extreme values of $G$ for $\alpha$ and $\beta$ are

$$\alpha = \frac{1}{2(T-1)} - \beta \qquad\qquad \beta = \frac{1}{2T},$$

which both have positive second derivative. Then, the condition $G \geq 0$ translates into the following condition on $T, E$:

$$\frac{1}{T-1} + 4(T-1) - \frac{E}{T} \geq 0. \tag{5.2}$$

**open facet $\alpha = 0$.** In this case the extreme $\beta$ value for $G$ is

$$\beta = \frac{E + \gamma E + 2\gamma T - 2\gamma ET}{2T(E-1)}.$$

However, then the second order derivative of $G$ in $\gamma$ is

$$\frac{\partial^2 G}{\partial^2 \gamma} = -\frac{E^2}{2T(E-1)},$$

which is negative. Hence, there is no local minimum in this open facet.

**open facet $\beta = 0$.** The extreme $\alpha$ and $\gamma$ values for $G$ are

$$\alpha = \frac{1 + 3\gamma - 2\gamma T}{2(T-1)} \qquad \gamma = \frac{(2-T)(2T-1)}{4E(T-1^2) - T(5-4T(2-T))}.$$

In the considered region for $(T, E)$ the value of $\alpha + \gamma$ exceeds 1, and is therefore outside the boundaries of the triangle.

**edge $(\alpha, \beta, \gamma) = (x, 1-x, 0)$ for $0 \leq x \leq 1$.** The extreme point for $x$ is

$$x = 1 - \frac{1}{2T}$$

and it is a local minimum. For this point the condition $G \geq 0$ translates into the condition

$$T(T-1) - \frac{3}{4} - \frac{E}{4T} \geq 0. \tag{5.3}$$

**edge $(\alpha, \beta, \gamma) = (x, 0, 1-x)$ for $0 \leq x \leq 1$.** The extreme point

$$x = \frac{2ET + 2T - 2T^2 - 2E - 1}{2(E-T)(T-1)},$$

is a local minimum. It generates the condition

$$4E(1-(2-T)T^2) - (2T(T-1)-1)^2 \geq 0. \tag{5.4}$$

**edge $(\alpha, \beta, \gamma) = (0, x, 1-x)$ for $0 \leq x \leq 1$.** Here, $G$ is linearly increasing in $x$. Hence, a local minimum is reached at $x = 0$, generating the condition

$$E(T-1) - 2 \geq 0. \tag{5.5}$$

**edge** $(\alpha, \beta, \gamma) = (x, 0, 0)$ **for** $0 \leq x \leq 1$**.** The extreme point for $x$ is

$$x = \frac{1}{2(T-1)},$$

which is a local minimum and generates the condition

$$4T - 5 - \frac{1}{T-1} \geq 0. \tag{5.6}$$

**edge** $(\alpha, \beta, \gamma) = (0, x, 0)$ **for** $0 \leq x \leq 1$**.** The extreme point for $x$ is

$$x = \frac{E}{2T(E-1)},$$

which is a local minimum and generates the condition

$$4(T-1) - \frac{E^2}{T(E-1)} \geq 0. \tag{5.7}$$

**edge** $(\alpha, \beta, \gamma) = (0, 0, x)$ **for** $0 \leq x \leq 1$**.** The extreme point

$$x = \frac{1}{2(ET - E - T)},$$

is a local minimum. It generates the condition

$$T - 1 - \frac{1}{4(ET - E - T)} \geq 0. \tag{5.8}$$

The parameters $T$ and $E$ describe the algorithm and thus we want to find values $T, E$ that satisfy all Conditions (5.1) to (5.8) and minimize $T$. In the considered region for $T$ and $E$, the conditions (5.3) and (5.5) to (5.7) are always satisfied. Hence, we focus on the remaining conditions. The optimal point lies on the intersection of the left hand sides of Conditions (5.2) and (5.4). Then one can compute that choosing $E = T(3 - 8T + 4T^2)/(T-1)$ and $T$ as the root to the following polynomials of degree 5 in the interval $T \in [1.5, 2]$ is optimal:

$$-1 - 16T + 20T^2 + 36T^3 - 52T^4 + 16T^5.$$

Numerically, we obtain the optimal parameters $T \approx 1.7453$ and $E \approx 2.8609$. □

### 5.3.2 Lower Bound for Randomized Algorithms

In this section we give a lower bound on the best possible competitive ratio of any randomized algorithm against an oblivious adversary. We do so by specifying a probability

distribution over inputs and proving a lower bound on $E[ALG]/E[OPT]$ that holds for all deterministic algorithms $ALG$. By Yao's principle [Yao77, BEY98] this gives the desired lower bound. Note that we use a different variant of Yao's principle than in Section 4.6 in Chapter 4, where we consider $E[ALG/OPT]$.

The probability distribution over inputs with $n$ jobs has a constant parameter $0 < q < 1$ and is defined as follows: Each job $j$ has upper limit $\bar{p}_j = 1/q > 1$, and its processing time $p_j$ is set to 0 with probability $q$ and to $1/q$ with probability $1 - q$.

First, observe that we only need to consider algorithms that schedule a job $j$ immediately if the job has been tested and $p_j = 0$. Furthermore, we only need to consider algorithms that never create idle time before all jobs are completed. We claim that any such algorithm satisfies $E[ALG] \geq n^2/(2q)$ for all $n$.

Lemma 5.13 *Any algorithm that schedules a job $j$ immediately if the job has been tested and $p_j = 0$ and never creates idle time before all jobs are completed satisfies $E[ALG] \geq n^2/(2q)$ for all $n$.*

*Proof.* We prove this by induction on $n$. Let $ALG(k)$ denote the objective value of the algorithm $ALG$ executed for a random instance with $k$ jobs that is generated by our probability distribution for $n = k$ (i.e., all $k$ jobs have $\bar{p}_j = 1/q$ and $p_j$ is set to 0 with probability $q$ and to $1/q$ otherwise).

Consider the base case $n = 1$. If $ALG$ executes job 1 without testing, then $ALG(1) = 1/q$. If $ALG$ tests the job and then necessarily executes it right away, since there are no other jobs, then $E[ALG(1)] = 1 + (q \cdot 0 + (1 - q) \cdot (1/q)) = 1/q$. In both cases, $E[ALG(1)] = 1/q \geq n^2/(2q)$.

Now assume the claim has been shown for $n-1$, i.e., $E[ALG(n-1)] \geq (n-1)^2/(2q)$. Consider the execution of $ALG$ on an instance with $n$ jobs, and distinguish how the algorithm handles the first job. Without loss of generality, assume that this job is job 1.

**Case 1:** $ALG$ executes job 1 without testing (completing at time $C_1 = 1/q$), or it tests jobs 1 and then executes it immediately independent of its processing time. In the latter case job 1 has expected completion time $E[C_1] = 1 + (1 - q)/q = 1/q$. After the completion of job 1, the algorithm schedules the remaining $n - 1$ jobs, which is a random instance with $n - 1$ jobs. Hence, the objective value is

$$\begin{aligned} E[C_1] + E[C_1](n-1) + E[ALG(n-1)] &= 1/q + (n-1)/q + E[ALG(n-1)] \\ &\geq n/q + n^2/(2q) - n/q = n^2/(2q). \end{aligned}$$

**Case 2** *ALG* tests job 1 and then executes it immediately if its processing time is 0, but defers it if its processing time is $1/q$. Assume first, that if $p_1 = 1/q$ then *ALG* defers the execution of $p_1$ to the very end of the schedule. We have

$$E[ALG(n)\big|p_1 = 0] = 1 + (n-1) + E[ALG(n-1)]$$

and

$$E[ALG(n)\big|p_1 = 1/q] = n + E[ALG(n-1)] + E[len(ALG(n-1))] + 1/q,$$

where $len(ALG(n-1))$ is the length of the schedule for $n-1$ jobs. Note that every job contributes $1/q$ to the expected schedule length no matter whether it is tested (in which case it requires time 1 for testing and an additional expected $(1-q)/q$ time for processing) or not (in which case its processing time is $1/q$ for sure). Therefore, $E[len(ALG(n-1))] = (n-1)/q$. So we have:

$$\begin{aligned} E[ALG(n)] &= q \cdot \Big(n + E[ALG(n-1)]\Big) + (1-q)\Big(n + n/q + E[ALG(n-1)]\Big) \\ &= qn + n + n/q - qn - n + E[ALG(n-1)] \\ &= n/q + E[ALG(n-1)] \\ &\geq n^2/(2q). \end{aligned}$$

Finally, we need to consider the possibility that $p_1 = 1/q$ and *ALG* defers job 1, but schedules it at some point during the schedule for the remaining $n-1$ jobs instead of at the very end of the schedule. Assume that *ALG* schedules job 1 in such a way that $k$ of the remaining $n-1$ jobs are executed after job 1. We compare this schedule to the schedule where job 1 is executed at the very end of the schedule. Let $K$ be the set of the $k$ jobs that are executed after job 1 by *ALG*. Note that the jobs in the set $K$ can be jobs that are scheduled without testing (and thus executed with processing time $1/q$), jobs that are tested and executed after the execution of job 1 (so that the expected time for testing and executing them is $1/q$), or jobs that are tested before the execution of job 1 but executed afterwards (in which case their processing time must be $1/q$, since jobs with processing time 0 are executed immediately after they are tested). Hence, moving the execution of job 1 from the very end of the schedule ahead of $k$ job executions will change the expected objective value as follows: The expected completion time of job 1 decreases by $k/q$, and the completion time of each of the $k$ jobs in $K$ increases by $1/q$. Therefore, $E[ALG(n)]$ is the same as when job 1 is executed at the end of the schedule, and we get $E[ALG(n)] \geq n^2/(2q)$ as before. □

Theorem 5.14 *There is no randomized algorithm with competitive ratio* $c < 3/23 \cdot \left(9 + 2\sqrt{3}\right) \approx 1.6257$.

*Proof.* We first consider the cost of the optimal schedule. Let $Z$ denote the number of jobs with processing time 0. Note that $Z$ is a random variable with binomial distribution. The optimal schedule first tests and executes the $Z$ jobs with $p_j = 0$ and then executes the $n - Z$ jobs with $p_j = 1/q$ untested. Hence, the objective value of $OPT$ is:

$$\frac{Z(Z+1)}{2} + Z(n-Z) + \frac{(n-Z)(n-Z+1)}{2q}.$$

Using $E[Z] = nq$ and $E[Z^2] = (nq)^2 + nq(1-q)$, we obtain

$$E[OPT] = \frac{n^2}{2}\left(\frac{1}{q} + 3q - 2 - q^2\right) + O(n).$$

Since we have $E[ALG] \geq n^2/(2q)$, we obtain a lower bound for randomized algorithms against an oblivious adversary by applying Yao's principle [Yao77, BEY98]. In the limit for $n \to \infty$ this is

$$\frac{1/q}{1/q + 3q - 2 - q^2}.$$

The expression attains its maximum for the parameter $q = 1 - 1/\sqrt{3} \approx 0.4226$. Then, this yields a lower bound of $3/23 \cdot \left(9 + 2\sqrt{3}\right) \approx 1.6257$. □

## 5.4 Deterministic Algorithms for Uniform Upper Limits

One of the striking questions for scheduling with testing is if there is a deterministic algorithm with performance better than 2. We present two algorithms that achieve this for special instance classes. First we consider instances with uniform upper limit. We combine our algorithm Threshold with a new algorithm Beat that cleverly handles instances with upper limit roughly 2 and show this yields competitive ratio 1.9338. Our second algorithm, Ute is for extreme uniform instances. They have a uniform upper limit and all processing times are either 0 or the upper limit. We use these instances in the construction of the deterministic lower bound and Ute is nearly tight on this lower bound instance. In general, on extreme uniform instances we prove Ute has competitive ratio 1.8668.

### 5.4.1 An Improved Algorithm for Uniform Upper Limits

In this section we present an algorithm for instances with uniform upper limit $\bar{p}$ that achieves a ratio strictly less than 2. We present a new algorithm Beat that performs well on instances with upper limit roughly 2, but its performance becomes worse for larger upper limits. Thus, in this case we employ the algorithm Threshold presented in Section 5.2.1.

To simplify the analysis, we consider the limit of $ALG(I)/OPT(I)$ when the number of jobs $n$ approaches infinity. We say that an algorithm $ALG$ is *asymptotically* $\rho_\infty$*-competitive* or *has asymptotic competitive ratio at most* $\rho_\infty$ if the following holds:

$$\limsup_{n\to\infty} \sup_{I} \frac{ALG(I)}{OPT(I)} \leq \rho_\infty.$$

Algorithm (Beat) The algorithm Beat balances the time testing jobs and the time executing jobs while there are untested jobs. A job is called *short* if its running time is at most $E = \max\{1, \bar{p} - 1\}$, and *long* otherwise. Let TotalTest denote the time we spend testing long jobs and let TotalExec be the time long jobs are executed. We iterate testing an arbitrary job and then execute the job with smallest processing time either, if it is a short job, or if TotalExec + $p_k$ is at most TotalTest. Once all jobs have been tested, we execute the remaining jobs in order of non-decreasing processing time.

We analyze the performance of Beat below and show that the adversary gives jobs with $p_j \in \{0, E, \bar{p}\}$ and at most one job with $p_j \in (E, \bar{p})$ in order of decreasing $p_j$ in a worst-case instance. This enhances our understanding of the structure of the schedules produced by Beat and $OPT$. We will prove that the asymptotic competitive ratio of Beat for $\bar{p} \leq 3$ is at most

$$\rho_\infty^{BEAT} = \frac{1 + 2(-2 + \bar{p})\bar{p} + \sqrt{(1 - 2\bar{p})^2(-3 + 4\bar{p})}}{2(-1 + \bar{p})\bar{p}}.$$

This function decreases, when $\bar{p}$ increases. Alternatively, for small upper limit we can execute each job without test. Then there is a worst-case instance where all jobs have processing time $p_j = 0$. The optimal schedule tests each job only if the upper limit $\bar{p}$ is larger than one and executes it immediately. For $\bar{p} < 1$ this means the competitive ratio is 1 and otherwise it is $\bar{p}$, which monotonously increases. Thus, we choose a threshold $T_1 \approx 1.9338$ for $\bar{p}$, where we start applying Beat: the fixpoint of the function $\rho_\infty^{BEAT}$.

For some upper limit $\bar{p} > 3$ the performance behavior of Beat changes and the asymptotic competitive ratio increases in our analysis below. Thus, we employ the

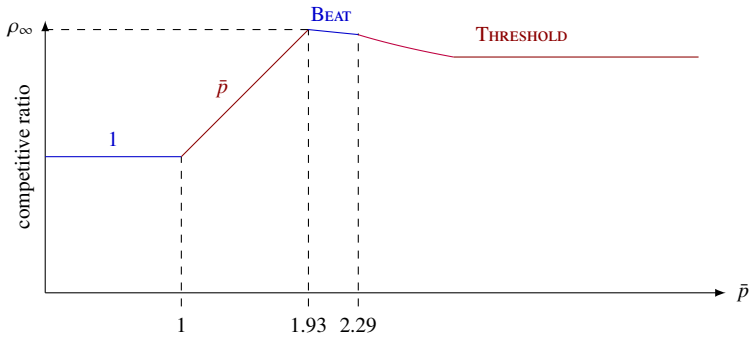



Figure 5.9: Competitive ratio depending on $\bar{p}$.

algorithm THRESHOLD from Section 5.2.1 for larger upper limits. Recall that for $\bar{p} > 2$ THRESHOLD tests all jobs, executes those with $p_j \leq 2$ immediately and defers the other jobs. We argue that there is a worst-case instance with short jobs that have processing time 0 or 2 and long jobs with processing time $\bar{p}_j = \bar{p}$ and that no long job is tested in an optimal solution. This will allow us to prove

$$\rho_\infty^{THRESH} = \begin{cases} \frac{-3+\bar{p}+\sqrt{-15+\bar{p}(18+\bar{p})}}{2(\bar{p}-1)} & \text{if } \bar{p} \in (2,3) \\ \sqrt{3} \approx 1.73 & \text{if } \bar{p} \geq 3 \end{cases}.$$

The function for small $\bar{p}$ is a monotone function decreasing from 2 to $\sqrt{3}$ in the limits for $\bar{p} \in (2,3)$. We choose the threshold, where we change from applying BEAT to employing THRESHOLD at $T_2 \approx 2.2948$, the crossing point of the two functions $\rho_\infty^{BEAT}$ and $\rho_\infty^{THRESH}$ describing the competitive ratio of BEAT and THRESHOLD in the interval $(2,3)$.

**Algorithm** Execute all jobs without test, if the upper limit $\bar{p}$ is less than $T_1 \approx 1.9338$. Otherwise, if the upper limit $\bar{p}$ is greater than $T_2 \approx 2.2948$, execute the algorithm THRESHOLD. For all upper limits between $T_1$ and $T_2$, execute the algorithm BEAT.

The function describing the asymptotic competitive ratio depending on $\bar{p}$ is displayed in Figure 5.9. Its maximum is attained at $T_1$, which is a fixpoint.

**Theorem 5.15** *The asymptotic competitive ratio of our algorithm is $\rho_\infty = T_1 \approx 1.9338$, which is the only real root of $2\bar{p}^3 - 4\bar{p}^2 + 4\bar{p} - 1 - \sqrt{(1-2\bar{p})^2(4\bar{p}-3)}$.*

### Analysis of BEAT

The algorithm BEAT balances the time testing jobs and the time executing jobs while there are untested jobs. A job is called *short* if its running time is at most $E = \max\{1, \bar{p}-$

1}, and *long* otherwise. Let TotalTest denote the time we spend testing long jobs and let TotalExec be the time long jobs are executed. We iterate testing an arbitrary job and then execute the job with smallest processing time either, if it is a short job, or if TotalExec $+p_k$ is at most TotalTest. We call this the first part of the schedule. Once all jobs have been tested, we execute the remaining jobs in order of non-decreasing processing time. This is the second part of the schedule. The pseudo code is shown in Algorithm 5.1.

**Algorithm 5.1:** BEAT

**Input:** A set of $n$ jobs with uniform upper limit $\bar{p}$.

**Output:** A schedule of tests and executions of all jobs.

```
1  TotalTest ← 0; // total time of executed tests
2  TotalExec ← 0; // total time of executed jobs
3  while there are untested jobs do
4      k ← tested, not executed job with minimum p_k; // p_k = ∞ if no such
         job
5      if TotalExec + p_k ≤ TotalTest then
6          Execute k;
7          TotalExec ← TotalExec + p_k;
8      else
9          j ← untested job with minimum p̄_j;
10         Test j;
11         if p_j ≤ E then
12             Execute j;
13         else
14             TotalTest ← TotalTest + 1;
15 Execute all remaining jobs in order of non-decreasing p_j;
```

We make a structural observation about the algorithm schedule for a worst-case instance.

Lemma 5.16 *The adversary gives jobs with $p_j \in \{0, E, \bar{p}\}$ and at most one job with $p_j \in (E, \bar{p})$ in order of decreasing processing time $p_j$.*

*Proof.* We first consider the ordering of test results. For a fixed number of short and long jobs, moving the test result of a short job towards the end of the algorithm does not

affect the optimal cost. For the algorithm cost, the test of a short job can either move behind the test of a long and delayed job or behind the test and execution of a long job. In the first case, the algorithm cost increases by 1, in the second case it increases by the processing time difference between the long and the short job, which is non-negative. Thus, it increases in both cases, which means short jobs are the last test results in an adversarial sequence.

Long jobs have processing time larger than $E \geq \bar{p} - 1$, which means they are not tested in the optimal schedule. Hence, increasing their processing time does not increase the optimal cost. For the delayed jobs, increasing their processing time to $\bar{p}$ increases the algorithm cost, but does not change the schedule, so in an adversarial sequence all delayed jobs have $p_j = \bar{p}$. For the executed jobs, note that no two jobs are executed without a test in between, as their processing time is larger than one, the length of a test. Thus we can assume they are each tested immediately before their execution. An adversarial sequence presents them ordered by decreasing processing time. We want to show they all have processing time $\bar{p}$. For this, we use the following iterative procedure: While the last long and executed job is followed by the test of a long job, we increase its processing time until either $p_j = \bar{p}$ or it is followed by the test of a short job. Then we reorder the executed jobs by decreasing processing time. Otherwise, if there is more than one executed job with $p_j < \bar{p}$, we shift processing time from the last long executed job to the one before. This increases the completion time of the first of the two jobs but does not change the completion time of any other job. Once the last long executed job's processing time decreases to $E$ it becomes a short job and does not change the algorithm schedule. We repeat these steps until there is at most one long and executed job with $p_j < \bar{p}$, the last one, and it is followed by tests of short jobs.

Finally, we observe that in the algorithm and in the optimal schedule all short jobs with $p_j \in [0, \bar{p} - 1]$ are tested independent of their actual processing time. Also, the execution order of the algorithm and the optimal schedule solely depend on the ordering of the processing times. Therefore, Lemma 5.3 implies that short jobs have processing times either 0 or $\bar{p} - 1$. To conclude, we can assume without loss of generality that all short jobs with processing time at least $\bar{p} - 1$ are not tested in the optimal solution. Then, increasing their processing time to $E$ does not change the optimal cost. It increases the algorithm cost, so in a worst-case adversarial sequence all short jobs have $p_j \in \{0, E\}$. □

Consequently, the schedule produced by Beat consists of the following parts, see

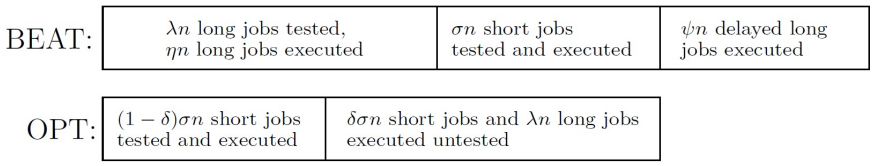



Figure 5.10: Structure of schedules produced by BEAT and OPT.

also Figure 5.10:

- The tests of the $\lambda$ fraction of jobs, that are long jobs, interleaved with executions of the $\eta$ fraction of jobs, that are also long jobs and that are executed during the "while there are untested jobs" loop.
- The tests and immediate executions of the short jobs, which is a $\sigma = 1-\lambda$ fraction of all jobs. Let $\delta$ be the fraction of short jobs with $p_j = E$.
- The executions of the $\psi = \lambda - \eta$ fraction of jobs, that are delayed long jobs, in the "execute all remaining jobs" statement.

OPT consists of the following parts (in this order), see also Figure 5.10:

- The tests and immediate executions of the $(1-\delta)\sigma$ fraction of jobs that are short with $p_j = 0$.
- The untested executions of the $\delta\sigma$ fraction of jobs which are short and have processing time $E$ and the $\lambda$ fraction of jobs that are long.

We note that TotalTest has value $\lambda n$ when all long jobs are tested, so the execution time of long jobs in the first part, which is at least $\bar{p}(n\eta - 1) + E$ by Lemma 5.16, cannot exceed $\lambda n$. As long jobs have $p_j > E \geq 1$, there are always as least as many long jobs tested as are executed. Thus, TotalExec never decreases below TotalTest $- \bar{p}$, as then some job can be executed. Hence, we have

$$\bar{p}\eta \leq \lambda + O(1/n) < \bar{p}\eta + O(1/n). \tag{5.9}$$

Furthermore, we have $\lambda = \eta + \psi$, which yields

$$\psi \leq (1 - 1/\bar{p})\,\lambda + O(1/n). \tag{5.10}$$

We first consider the algorithm schedule.

Lemma 5.17 *For a fraction $\delta \in [0,1]$ of short jobs with processing time $p_j = E$, we can bound the algorithm cost by*

$$ALG \leq \frac{n^2}{2} \cdot \left[\lambda^2\left(\bar{p} + 2 - \frac{1}{\bar{p}}\right) + \sigma^2((1+E)(2\delta - \delta^2) + (1-\delta)^2) + 2\lambda\sigma\left(2 + \left(1 - \frac{1}{\bar{p}}\right)(1 + E\delta)\right)\right] + O(n).$$

*Proof.* There is an $\eta$ fraction of long jobs completed in the first part of the algorithm schedule, each executed, when TotalExec$+p_j \leq$ TotalTest in the algorithm. Thus, the completion time of the $i$-th such job is at most $2i\bar{p} + 1$. The sum of these completion times is $\bar{p}\eta^2 n^2 + O(n)$. A fraction of $\delta\sigma$ jobs is short and has $p_j = E$. They are executed before the other $(1-\delta)\sigma$ fraction of jobs with $p_j = 0$ is executed. This means the completion times of the short jobs contribute

$$\begin{aligned} &n^2/2 \cdot \left[(1+E)\delta^2\sigma^2 + (1-\delta)^2\sigma^2 + 2(1+E)\delta\sigma(1-\delta)\sigma\right] + O(n) \\ &= n^2/2 \cdot \left[\sigma^2((1+E)(2\delta - \delta^2) + (1-\delta)^2)\right] + O(n). \end{aligned}$$

Additionally there is an $\psi$ fraction of jobs, which are executed at the end of the schedule, each with processing time $\bar{p}$. Thus their contribution to the algorithm cost is $\bar{p}\psi^2 n^2/2 + O(n)$. The execution of the fraction $\sigma$ of short jobs starts latest at time $n\lambda + \bar{p}n\eta$, and the execution of the fraction $\psi$ of jobs is delayed by at most $n\lambda + \bar{p}n\eta + (1+E\delta)n\sigma$. Thus, the total objective value of Beat is at most:

$$ALG \leq n^2/2 \cdot \left[2\bar{p}\eta^2 + \sigma^2((1+E)(2\delta - \delta^2) + (1-\delta)^2) + \bar{p}\psi^2 + 2(\lambda + \bar{p}\eta)\sigma + 2(\lambda + \bar{p}\eta + (1+E\delta)\sigma)\psi\right] + O(n).$$

By Equations (5.9) and (5.10), we know that $\eta \leq \lambda/\bar{p} + O(1/n)$ and $\psi \leq (1 - 1/\bar{p})\,\lambda + O(1/n)$. Together with $\eta + \psi = \lambda$, this yields the desired bound. □

Lemma 5.18 *For uniform upper limit $\bar{p} \in [1.5, 3]$, the asymptotic competitive ratio of* Beat *is at most*

$$\frac{1 + 2(-2+\bar{p})\bar{p} + \sqrt{(1-2\bar{p})^2(-3+4\bar{p})}}{2(-1+\bar{p})\bar{p}}.$$

*Proof.* We bounded the algorithm cost in Lemma 5.17 and thus first consider the optimal cost. In OPT, first a fraction $(1-\delta)\sigma$ of the short jobs is tested and executed with processing time 0. Then the remaining fraction $\delta\sigma$ of short jobs is executed with

processing time $\bar{p}$ without test. Thus their contribution to the sum of completion times is

$$n^2/2 \cdot \left[\sigma^2\left((1-\delta)^2 + \bar{p}\delta^2 + 2\delta(1-\delta)\right)\right] + O(n) \;= n/2 \cdot \left[\sigma^2\left((\bar{p}-1)\delta^2 + 1\right)\right] + O(n).$$

All long jobs are executed untested at the end of the schedule and take $\bar{p}$ time units. Their sum of completion times is $\bar{p}\lambda^2 n^2/2 + O(n)$ and they are each delayed by $\sigma n(1 + (\bar{p}-1)\delta))$, giving:

$$OPT = n^2/2 \cdot \left[\lambda^2\bar{p} + \sigma^2\left((\bar{p}-1)\delta^2 + 1\right) + 2\lambda\sigma\left(1 + (\bar{p}-1)\delta\right)\right] + O(n).$$

Then, the asymptotic competitive ratio $\rho_\infty$ for upper limit $\bar{p}$ in $[1.5, 3]$ is

$$\rho_\infty^{BEAT} = \frac{\lambda^2\left(\bar{p} + 2 - \frac{1}{\bar{p}}\right) + \sigma^2\left((1+E)(2\delta-\delta^2) + (1-\delta)^2\right) + 2\lambda\sigma\left(2 + \left(1-\frac{1}{\bar{p}}\right)(1+E\delta)\right)}{\bar{p}\lambda^2 + \sigma^2\left((\bar{p}-1)\delta^2 + 1\right) + 2\lambda\sigma\left(1 + (\bar{p}-1)\delta\right)}.$$

For $\sigma = 0$ or $\lambda = 0$ this fulfills the claim. For the other values we set $\sigma = \alpha\lambda$ so the ratio becomes:

$$\frac{\bar{p} + 2 - \frac{1}{\bar{p}} + \alpha^2\left((1+E)(2\delta-\delta^2) + (1-\delta)^2\right) + 2\alpha\left(2 + \left(1-\frac{1}{\bar{p}}\right)(1+E\delta)\right)}{\bar{p} + \alpha^2\left((\bar{p}-1)\delta^2 + 1\right) + 2\alpha\left(1 + (\bar{p}-1)\delta\right)}.$$

We take the term to mathematica to bound it. For the case $1.5 < \bar{p} < 2$ we show that the adversary chooses $\delta = 0$ and $\alpha$ such that the first derivative in $\alpha$ equals 0. Otherwise, in the case $2 \le \bar{p} \le 3$, we show for $\delta = 0$ that we get exactly the same expression as for $\bar{p} < 2$. We prove the adversary chooses this case, which means the competitive ratio is bounded by the following function

$$\rho_\infty^{BEAT} \le \frac{1 + 2(-2+\bar{p})\bar{p} + \sqrt{(1-2\bar{p})^2(-3+4\bar{p})}}{2(-1+\bar{p})\bar{p}}.$$

□

## 5.4.2 Analysis of Threshold for Uniform Upper Limits

In this section we analyze Algorithm Threshold (see Section 5.2.1) for instances with uniform upper limit $\bar{p} > 2$ and derive a competitive ratio as a function of $\bar{p}$.

Recall that for $\bar{p} > 2$, Threshold tests all jobs. It executes a job immediately if $p_j \le 2$, and defers it otherwise. We have proved in Lemma 5.4 that we may assume that all jobs with $p_j \le 2$ have execution times either 0 or 2. We also argued that in a worst case, Threshold first tests all long jobs, i.e., jobs $j$ with $p_j > 2$, then follow the short jobs with tests (first length-2 jobs and then length-0 jobs), and finally Threshold executes the deferred long jobs in increasing order of processing times.

An optimum solution tests a job $j$ only if $p_j + 1 < \bar{p}$. We show next that such long jobs to be tested in an optimal solution do not exist.

Lemma 5.19 *There is a worst-case instance with short jobs that have processing times* 0 *or* 2 *and long jobs with processing time* $p_j = \bar{p}$*. Furthermore, none of the long jobs is tested in an optimal solution.*

*Proof.* Consider an instance with short jobs with processing times 0 or 2 (Lemma 5.4). We may increase the processing time of untested long jobs to their upper limit $\bar{p}$ without changing the optimal schedule. This cannot decrease the worst-case ratio as the algorithm's objective value can only increase.

It remains to consider the long jobs that are tested by an optimal solution. We show that we may assume that those do not exist. This is trivially true if $2 < \bar{p} < 3$. Then testing a long job $j$ costs $1 + p_j > 3$ which is greater than running the job untested at $\bar{p} < 3$, and thus, an optimal solution would never test it.

Assume now that $\bar{p} \geq 3$. Threshold schedules any long job *after* all short jobs; first it runs long tested jobs with total execution time $1 + p_j < \bar{p}$ in non-decreasing order of $p_j$ and then the untested jobs with execution time $\bar{p}$. As all untested jobs have processing time $p_j = \bar{p}$, we may assume that the algorithm and the optimum schedule long jobs in the same order. Reducing the processing times of all tested long jobs to $2 + \varepsilon$ for infinitesimally small $\varepsilon > 0$ does not change the two schedules and thus, by Proposition 5.2, the ratio of the objective values of the algorithm and the optimum does not decrease.

Now, we argue that reducing the processing times of tested long jobs from $2 + \varepsilon$ to 2 (thus making them short jobs) does not affect the optimal objective value, because $\varepsilon$ is infinitesimally small, and can only increase the objective value of the algorithm. Consider the first long job that is tested by the optimum and the algorithm, say job $\ell$. Consider the worst-case schedule of our algorithm for the new instance in which $\ell$ is turned into a short job with effectively the same processing time. Job $\ell$ is tested and scheduled just before the short jobs with $p_j = 0$ instead of after them. Let $a$ be the number of those short jobs. Then this change in $p_\ell$ to 2 improves the completion time of job $\ell$ by $a$ and increases the completion time of $a$ jobs by 2, so the net change in the objective value of the algorithm is $2a - a = a \geq 0$. The argument can be repeated until no tested long jobs are left. □

**Theorem 5.20** *For uniform upper limit $\bar{p} > 2$, Algorithm* Threshold *has an asymptotic competitive ratio at most*

$$\rho_{\infty}^{THRESH} = \begin{cases} \frac{-3+\bar{p}+\sqrt{-15+\bar{p}(18+\bar{p})}}{2(\bar{p}-1)} & \text{if } \bar{p} \in (2,3) \\ \sqrt{3} \approx 1.73 & \text{if } \bar{p} \geq 3 \end{cases}.$$

*The function for small $\bar{p}$ is a monotone function decreasing from 2 to $\sqrt{3}$ in the limits for $\bar{p} \in (2,3)$.*

*Proof.* Consider a worst-case instance according to Lemma 5.19. Let $\alpha n$ denote the number of short jobs of length 0, let $\beta n$ be the number of short jobs of length 2, and let $\gamma n$ be the number of long jobs with $p_j = \bar{p}$. There are no other jobs, so $\alpha + \beta + \gamma = 1$. Recall, that we may assume that Threshold's schedule is as follows: first $\gamma n$ tests, $\beta n$ tests and executions of length-2 jobs, then tests and executions of $\alpha n$ length-0 jobs, followed by the execution of long jobs with $p_j = \bar{p}$. The objective value is

$$ALG = n^2/2 \cdot \left[2\gamma(\alpha+\beta+\gamma) + \beta^2 \cdot 3 + 2 \cdot 3\beta(\alpha+\gamma) + \alpha^2 + 2\alpha\gamma + \gamma^2 \cdot \bar{p}\right] + O(n). \tag{5.11}$$

To estimate the objective value of an optimal solution, we distinguish two cases for the upper limit $\bar{p}$.

**Case: $\bar{p} > 3$.** In this case, an optimal solution would test all short jobs, first the length-0 jobs and then the length-2 jobs. Then all long jobs follow without testing them (Lemma 5.19). Using the above notation, we have an optimal objective value

$$OPT = n^2/2 \cdot \left[\alpha^2 + 2\alpha(\beta+\gamma) + \beta^2 \cdot 3 + 2 \cdot 3\beta\gamma + \gamma^2 \cdot \bar{p}\right] + O(n).$$

Using $\gamma = 1 - \alpha - \beta$, the asymptotic competitive ratio for any $\bar{p} > 3$ can be bounded by

$$\frac{2 - \alpha^2 - 2\alpha\beta + (4-3\beta)\beta + \bar{p}(-1+\alpha+\beta)^2}{-\alpha^2 + \alpha(2-6\beta) - 3(-2+\beta)\beta + \bar{p}\,(-1+\alpha+\beta)^2},$$

which has its maximum at $\sqrt{3}$ for $\alpha = \left(3 - \sqrt{3}\right)/2$ and $\beta = \left(\sqrt{3} - 1\right)/2$.

**Case: $\bar{p} \leq 3$.** In this case, an optimal solution tests only short jobs with $p_j = 0$ and executes all other jobs untested, also short jobs with $p_j = 2$. The value of an optimal schedule is

$$OPT = n^2/2 \cdot \left[\alpha^2 + 2\alpha(\beta+\gamma) + \bar{p} \cdot \beta^2 + 2\bar{p} \cdot \beta\gamma + \bar{p} \cdot \gamma^2\right] + O(n).$$

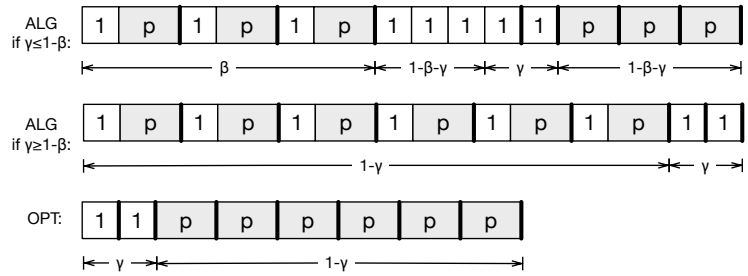



Figure 5.11: The schedule produced by UTE and the optimal schedule.

With the value of THRESHOLD's solution given by Equation (5.11), the asymptotic competitive ratio is

$$\rho_{\infty}^{THRESH} = \frac{\alpha^2 + 3\beta^2 + 8\beta\gamma + \alpha(6\beta + 4\gamma) + \gamma^2(2 + \bar{p})}{\alpha^2 + 2\alpha(\beta + \gamma) + (\beta + \gamma)^2\bar{p}}.$$

Using mathematica we verify that this ratio has its maximum at the desired value

$$\rho_{\infty}^{THRESH} \leq \frac{-3 + \bar{p} + \sqrt{-15 + \bar{p}(18 + \bar{p})}}{2(\bar{p} - 1)}.$$ □

### 5.4.3 Nearly Tight Algorithm for Extreme Uniform Instances

We present a deterministic algorithm for the restricted class of *extreme uniform* instances, that is almost tight for the instance that yields the deterministic lower bound. An *extreme uniform* instance consists of jobs with uniform upper limit $\bar{p}$ and processing times in $\{0, \bar{p}\}$. Our algorithm UTE attains asymptotic competitive ratio $\rho \approx 1.8668$ for this class of instances.

**Algorithm (UTE)** If the upper limit $\bar{p}$ is at most $\rho$, then all jobs are executed without test. Otherwise, all jobs are tested. The first $\max\{0, \beta\}$ fraction of the jobs are executed immediately after their test. The remaining fraction of the jobs are executed immediately after their test if they have processing time 0 and are delayed otherwise, see Figure 5.11. The parameter $\beta$ is defined as

$$\beta = \frac{1 - \bar{p} + \bar{p}^2 - \rho + 2\bar{p}\rho - \bar{p}^2\rho}{1 - \bar{p} + \bar{p}^2 - \rho + \bar{p}\rho}. \tag{5.12}$$

**Theorem 5.21** *The competitive ratio of* UTE *is at most* $\rho = \frac{1+\sqrt{3+2\sqrt{5}}}{2} \approx 1.8668$.

*Proof.* If the upper limit $\bar{p}$ is at most $\rho$, by Lemma 5.1 the algorithm has competitive ratio $\bar{p}$, which fulfills the claim. Thus, we assume in the following $\bar{p} \geq \rho$. An instance is defined by the job number $n$, an upper limit $\bar{p}$, and a fraction $\gamma$ such that the first $\gamma$ fraction of the jobs tested by Ute have processing time $\bar{p}$, while the jobs in the remaining $1 - \gamma$ fraction have processing time 0. Thus, the optimal cost is

$$OPT = n^2/2 \cdot \left[\gamma^2 + \bar{p}(\gamma - 1)^2 + 2\gamma(1 - \gamma)\right] + n/2 \cdot \left[\gamma + \bar{p}(1 - \gamma)\right].$$

The algorithm chooses $\beta$ so as to have the smallest ratio $\rho$. With the chosen fixed value of $\rho$, the value $\beta$ from Equation (5.12) is a decreasing function in $\bar{p}$ for $\bar{p} \geq \rho$. Hence, there is a threshold value $p^*$ such that $\beta(\bar{p}) \leq 0$ for all $\bar{p} \geq p^*$. For $\rho \approx 1.8668$ we have $p^* \approx 2.796$. We derive the best value $\beta$ for the algorithm by analyzing the competitive ratio for $n \to \infty$ and later consider the linear terms separately. For the algorithm cost, we distinguish three cases depending on the ranges of $\bar{p}$ and $\gamma$. The case that makes the analysis tight is $\rho \leq \bar{p} \leq p^*$ and $\gamma \leq 1 - \beta$, which we discuss next. The other cases are treated in the following Lemmas 5.22 and 5.23.

Consider for now $\beta$ and $\rho$ as some undetermined parameters which will be optimized in the analysis of this case. The schedule of Ute begins with a $\beta$ fraction of the jobs that are tested and executed with processing time $\bar{p}$ followed by a $\gamma$ fraction of the jobs that are tested and executed with processing time 0. The latter are delayed by $(1 - \gamma)$n tests and $\beta n$ executions of length $\bar{p}$. Last, there is a $(1 - \beta - \gamma)$ fraction of jobs delayed by $n$ tests and $\beta$ executions of length $\bar{p}$ and executed with processing time $\bar{p}$. Thus, we have

$$\begin{aligned} ALG = n^2/2 \cdot &\left[(\bar{p} + 1)\beta^2 + \gamma^2 + \bar{p}(1 - \beta - \gamma)^2 + 2(1 - \gamma + \bar{p}\beta)\gamma + 2(1 + \bar{p}\beta)(1 - \beta - \gamma)\right] \\ &+ n/2 \cdot \left[(\bar{p} + 1)\beta + \gamma + \bar{p}(1 - \beta - \gamma)\right]. \end{aligned}$$

The adversary chooses $\bar{p}$, the algorithm chooses $\beta$ depending on $\bar{p}$ and the adversary chooses $\gamma$ dependent on $\bar{p}$ and $\beta$. Thus, we can express the competitive ratio for this case as

$$\rho = \max_{\rho \leq \bar{p} \leq p^*} \min_{\beta} \max_{\gamma \leq 1-\beta} \lim_{n \to \infty} \frac{ALG}{OPT}.$$

Using mathematica, we show that the optimal choices are

$$\gamma = \frac{-\bar{p} + \beta\bar{p} - \rho + \bar{p}\rho}{(\rho - 1)(\bar{p} - 1)}, \qquad \beta = \frac{1 - \bar{p} + \bar{p}^2 - \rho + 2\bar{p}\rho - \bar{p}^2\rho}{1 - \bar{p} + \bar{p}^2 - \rho + \bar{p}\rho}, \qquad \bar{p} = \rho,$$

which explains our choice of $\beta$. Then, we have

$$\rho = \frac{1 + \sqrt{3 + 2\sqrt{5}}}{2} \approx 1.8668.$$

For the linear terms in $n$, it is easy to check for our choice of $\beta$ and $\rho$

$$\frac{(\bar{p}+1)\beta+\gamma+\bar{p}(1-\beta-\gamma)}{\gamma+\bar{p}(1-\gamma)} \leq \rho$$

holds for all $\gamma \in [0, 1-\beta]$ and $\rho \leq \bar{p} \leq p^*$. This completes the proof assuming the subsequent Lemmas 5.22 and 5.23 hold. □

**Lemma 5.22** *For $\bar{p} > p^*$* Ute *has competitive ratio $\rho \approx 1.8668$.*

*Proof.* In this case $\beta \leq 0$ and Ute first tests and postpones the first $1-\gamma$ fraction of jobs (all of length $\bar{p}$) and then tests and executes the remaining $\gamma$ fraction (all of length 0). Thus, the cost for the algorithm is

$$ALG = n^2/2 \cdot \left[\gamma^2 + \bar{p}(1-\gamma)^2 + 2(1-\gamma)\gamma + 2(1-\gamma)\right] + n/2 \cdot [\gamma + (\bar{p}+1)(1-\gamma)].$$

The ratio is at most $\rho$, if $g \geq 0$ for

$$\begin{aligned} g &:= 2/n \cdot (\rho OPT - ALG) \\ &= -3 + 2\gamma - 2n\gamma + n\gamma^2 + p(-1+\gamma)(-1-n+n\gamma)(-1+\rho) \quad + \gamma\rho + 2n\gamma\rho - n\gamma^2\rho \\ &\quad + \gamma^2(\bar{p}-1)(\rho-1) + \bar{p}(\rho-1) + 2\gamma(\bar{p}+\rho-\bar{p}\rho) - 1. \end{aligned}$$

The expression $g$ is increasing in $\bar{p}$ as its derivative is $(1+n(1-\gamma))(1-\gamma)(\rho-1) > 0$. Therefore we can assume for the worst case $\bar{p} = p^*$. Now we observe that $g$ is convex in $\gamma$ as the second derivative is $n\left(1+\sqrt{4\rho-3}\right) > 0$. Hence, the adversary chooses the extreme point for $g$ in $\gamma$, namely

$$\gamma = \frac{-1+\sqrt{4\rho-3}}{1+\sqrt{4\rho-3}}.$$

With these choices of $\bar{p}$ and $\gamma$ the expression $g$ has the form

$$g = \frac{3-2(2-\rho)\rho-\sqrt{4\rho-3}}{2(\rho-1)}.$$

We show $g$ is positive for $\rho \in [1.6, 2]$ and $n \geq 1$, which proves competitive ratio $\rho$ for this case. □

**Lemma 5.23** *For $\rho \leq \bar{p} \leq p^*$ and $\gamma \geq 1-\beta$ algorithm* Ute *has competitive ratio $\rho \approx 1.8668$.*

*Proof.* In this case the algorithm does not postpone the execution of jobs. The first $1-\gamma$ fraction of jobs have processing time $\bar{p}$ and the last $\gamma$ fraction jobs have processing time 0. Therefore the cost of Ute is

$$ALG = n^2/2 \cdot \left[(\bar{p}+1)(1-\gamma)^2 + \gamma^2 + 2(\bar{p}+1)\gamma(1-\gamma)\right] + n/2 \cdot [(\bar{p}+1)(1-\gamma)+\gamma].$$

We analyze the quadratic and the linear terms in $n$ separately. The ratio of the quadratic terms is at most $\rho$, if we have $g_2 \geq 0$ for

$$g_2 := 2/n^2 \cdot \lim_{n\to\infty} (\rho OPT - ALG) = -1 - (1-\gamma^2)\bar{p} + (\bar{p} - (2-\gamma)\gamma(\bar{p}-1))\rho.$$

The value of $\beta$ is maximized at $\bar{p} = \rho$, which is approximately $\beta^* := 0.2869$. We observe that $g_2$ is decreasing in $\gamma$ and thus consider $\gamma = 1-\beta$. For this choice $g_2$ is positive for $\bar{p} \in [1.7, p^*]$. Therefore, we have shown that the ratio is at most $\rho$ for the quadratic terms.

For the linear terms, the ratio is at most $\rho$, if $g_1 \geq 0$ for

$$g_1 := -1 + \gamma\rho + \bar{p}(-1+\gamma+\rho-\gamma\rho).$$

This is increasing in $\bar{p}$, and for $\bar{p} = \rho$ it is positive. □

Remark 5.24 The deterministic lower bound 1.8546 (Theorem 5.9 in Section 5.2.2) uses the upper limit $\bar{p} \approx 1.9896$. Plugging this choice of $\bar{p}$ into Equation (5.12) shows that Ute has asymptotic competitive ratio $\rho_\infty \approx 1.8552$ on this instance, which is almost tight.

## 5.5 Optimal Testing for Minimizing the Makespan

We consider scheduling with testing with the objective of minimizing the makespan, i.e., the completion time of the last job that is processed. This objective function is special, as the time each job runs on the machine has a linear contribution to the makespan. This yields that for any algorithm that treats each job independent of the position where it occurs in the schedule, there is a worst-case instance containing only a single job.

Lemma 5.25 *If an algorithm that treats each job independent of the position where it occurs in the schedule is $\rho$-competitive for one-job instances, it is $\rho$-competitive also for general instances.*

*Proof.* Let an instance $I$ with $n$ jobs $j_1, \ldots, j_n$ and an arbitrary algorithm as in the statement of the lemma be given. Then the makespan $ALG(I)$ equals the sum of the makespans, if we split the instance into one-job instances. By assumption, the algorithm is $\rho$-competitive for each one-job instance. Thus, we have

$$ALG(I) = \sum_{i=1}^{n} ALG(\{j_i\}) \leq \sum_{i=1}^{n} \rho \cdot OPT(\{j_i\}) = \rho \cdot OPT(I). \qquad \square$$

We apply Lemma 5.25 to describe a deterministic algorithm with competitive ratio $\rho = \varphi$, the golden ratio, and show this is best-possible.

Theorem 5.26 *Testing each job $j$ if and only if $\bar{p}_j > \varphi \approx 1.6180$ is an algorithm with competitive ratio $\rho = \varphi$, which is best-possible.*

*Proof.* By Lemma 5.25 we just need to consider an instance consisting of a single job. Let that job have upper limit $\bar{p}$ and processing time $p$. If the algorithm does not test the job, then $\bar{p} \leq \varphi$. If $\bar{p} \leq 1$, the optimal schedule also executes the job untested, and the competitive ratio is 1. If $\bar{p} > 1$, the makespan of the algorithm is $\bar{p} \leq \varphi$ and the optimal makespan is at least 1, because the optimal makespan is minimized if the job is tested in the optimal schedule and reveals $p = 0$. Thus, the ratio is at most $\varphi$.

If the algorithm tests the job, then its makespan is $1 + p$, while the optimal makespan is $\min\{\bar{p}, 1 + p\}$. In the worst case, the job has processing time $p = \bar{p}$. Then the ratio is $(1 + \bar{p})/\bar{p}$, which decreases when the upper limit $\bar{p}$ increases. Thus, it is at most $(1 + \varphi)/\varphi = \varphi$.

To show this is best-possible, consider an instance with a single job with upper limit $\varphi$. Any algorithm that does not test this job has competitive ratio at least $\varphi$, as the optimal makespan is 1 if the job has processing time 0. Any other algorithm tests the job. If the job has processing time $\varphi$, the competitive ratio is $(1 + \varphi)/\varphi = \varphi$. $\square$

This shows that there is an algorithm that approaches the optimal processing time up to a factor $\varphi$. However, it does not know the optimal job ordering. Therefore this is not a $\varphi$-approximation for the sum of completion times.

Next we consider randomized algorithms. We first show that no randomized algorithm can have competitive ratio $\rho < 4/3$.

Theorem 5.27 *No randomized algorithm has competitive ratio $\rho < 4/3$ for minimizing the makespan of an instance of scheduling with testing.*

*Proof.* We want to apply Yao's principle [Yao77] and give a randomized instance for which no deterministic algorithm is better than 4/3-competitive. Consider a one-job instance with $\bar{p} = 2$. Let the job have $p = 0$ and $p = 2$ each with probability 0.5. The deterministic algorithm that does not test the job has expected makespan 2 and the deterministic algorithm testing the job also has expected makespan 2. The expected optimal solution size is 3/2. Thus, the instance yields the desired bound. □

For minimizing the makespan, the order in which jobs are treated is irrelevant by Lemma 5.25. Thus, the only decision an algorithm has to take is whether to test a job. Consider a job with upper limit $\bar{p}$. We show that the algorithm that executes the job untested if $\bar{p} \leq 1$ and otherwise tests it with probability $1-1/(\bar{p}^2-\bar{p}+1)$ is best-possible.

**Theorem 5.28** *Our randomized algorithm testing each job with $\bar{p} > 1$ with probability $1 - 1/(\bar{p}^2 - \bar{p} + 1)$ has competitive ratio 4/3, which is best-possible.*

*Proof.* By Lemma 5.25 we just need to consider an instance consisting of a single job. If its upper limit $\bar{p}$ satisfies $\bar{p} \leq 1$, the algorithm executes the job untested, which is optimal. Therefore, assume for the rest of the proof that $\bar{p} > 1$.

Note that Proposition 5.2, which was stated in the context of minimizing the sum of completion times, holds also for single-job instances where the objective is the makespan, because for one job the two objectives are the same. If $0 < p < \bar{p} - 1$, we observe that the optimal makespan and the expected makespan of the algorithm depend linearly on $p$, so by Proposition 5.2 we can set $p$ to 0 or $\bar{p} - 1$ without decreasing the competitive ratio. Now, if $\bar{p} - 1 \leq p < \bar{p}$, observe that increasing $p$ to $\bar{p}$ increases the expected makespan of the algorithm but does not affect the optimum. Therefore, we can assume that $p \in \{0, \bar{p}\}$ in a worst-case instance.

Let us first consider the case $p = \bar{p}$. Then the optimal solution schedules this job without test. Thus, the ratio of algorithm length over optimal length is

$$\rho = \frac{E[ALG]}{OPT} = \left(1 - \frac{1}{\bar{p}^2 - \bar{p} + 1}\right)\frac{\bar{p}+1}{\bar{p}} + \frac{1}{\bar{p}^2 - \bar{p} + 1} = \frac{\bar{p}^2}{\bar{p}^2 - \bar{p} + 1}.$$

Otherwise, we have $p = 0$. Then we have

$$\rho = \frac{E[ALG]}{OPT} = \left(1 - \frac{1}{\bar{p}^2 - \bar{p} + 1}\right) + \frac{1}{\bar{p}^2 - \bar{p} + 1}\bar{p} = \frac{\bar{p}^2}{\bar{p}^2 - \bar{p} + 1}.$$

This function is maximized at $\bar{p} = 2$, which yields competitive ratio 4/3. □

# Conclusion

Uncertainty exploration is an emerging research field in combinatorial optimization with strong ties to robust and online optimization. This makes it theoretically interesting and the numerous practical applications give it additional relevance. The classical model describes the data uncertainty as intervals, whose exploration yields the exact value at a fixed cost and asks to minimize the exploration cost to identify an optimal solution. In this thesis we describe new algorithms and lower bounds for this classical model and demonstrate its applicability in practice. We study variants of the classical model and showcase their potential. Then we propose combining the solution quality with the exploration cost in a single objective function as a new promising model for uncertainty exploration. We conclude with discussing the most important open questions in the field and point to interesting directions for future research.

For minimum spanning tree under uncertainty in the classical model the deterministic problem complexity is well-understood and there is a polynomial time algorithm for the verification problem. We show that randomized algorithms can beat the deterministic performance bounds, but there remains a gap between lower bound 1.5 and upper bound 1.707 on the best-possible competitive ratio for randomized algorithms. The current greedy approach consecutively consider the edges, but the example proving the analysis is tight suggests that a more global approach yields potential for improvement.

For the use of uncertainty exploration in practice it is desirable to have more computational experiments for the variety of problems that have been studied theoretically. We describe a class of instances that the preprocessing solves optimally. However, it would be important to understand on which instances the randomized algorithm performs better than the deterministic ones and vice versa as well as determining general instance characteristics that influence the performance of the algorithms.

We fully characterize set systems with competitive ratios 1 and 2, but there is no understanding of other families of set systems with constant competitive ratio. Our

lower bound construction employs a notion of disjointness of a set system. We consider two inclusion-wise maximal sets for which no other maximal set lies in the union and contains the complete intersection. The largest cardinality of the symmetric difference of two such sets equals the lower bound on the competitive ratio. It would be interesting to find optimization problems that have a small disjointness. Furthermore, to understand if this is the correct measure of difficulty, algorithms for such set systems are necessary.

It is very surprising and might appear unsatisfactory, that the usually easy bipartite matching problem has unbounded competitive ratio in the uncertainty setting. The parametrization by the cardinality of the largest maximal set that was proposed by Erlebach et al. [EHK16] does not seem to capture the problem difficulty better for our multiplicative definition of competitive ratio. However, research using an additive competitive ratio seems more promising. Here, wide gaps remain for both, bipartite matching and knapsack with uncertain profit.

The geometric interpretation of linear programs under uncertainty yields a new perspective to study uncertainty exploration. For example, the question to decide if a linear program allows a realization where the optimal solution queries a single element, seems intriguing. Geometrically expressed this asks the following: Given a space of dimension $d$ and a family of cones that all originate at the origin, is there a $d-1$ dimensional hyperplane through the origin that does not intersect any of the cones?

There are several models to study trade-offs between the exploration cost and the solution quality. We study minimizing the exploration cost to find an $\alpha$-approximate solution and show first results when there is an upper limit on the number of consecutive queries and instead parallel queries are necessary. One could also consider a fixed maximum number of parallel queries and minimize the number of rounds until a solution can be identified. Goerigk et al. [GGI+15] propose to optimize the solution quality given a fixed cost budget for the exploration. All these models have only been studied for few optimization problems and a thorough understanding of their advantages, drawbacks, and relation to each other is desirable.

In this thesis uncertain data is described by intervals and either a query yields the exact data or a new, possibly smaller, uncertainty interval. An interesting variant is to allow fractional queries. Paying cost $x$ then decreases the interval size by $x \cdot A_e$ either on a chosen side or symmetrically. This allows the algorithm to reduce the interval incrementally by $\varepsilon$ fractions and if the algorithm chooses the same intervals as in the optimal solution, they will be reduced by the same amount. Similar to the classical model,

the crucial decision is which intervals to reduce and in which order. It appears, such fractional queries may describe a nice intermediate model to resemble an LP solution.

A key to the algorithmic results in this thesis and a limiting factor for our lower bounds is the understanding of the optimal solution. The verification problem, computing an optimal solution for a fixed realization, is usually solvable in polynomial time. However, we are not aware of any results that bound the size of the optimal solution for a fixed uncertainty graph, but arbitrary realization. The main question is, if it is NP-hard to determine the best-possible lower bound.

Beyond the classical approach to uncertainty exploration we propose to combine the exploration cost and the solution quality in a single objective function. Our adversarial model for scheduling with testing shows that this leads to interesting questions with many applications. An immediate open question from our results is whether it is possible to achieve competitive ratio below 2 for minimizing the sum of completion times with a deterministic algorithm for arbitrary instances. Further interesting directions for future work include the consideration of job-dependent test times or other scheduling problems such as parallel machine scheduling or flow shop problems. More generally, the study of problems with explorable uncertainty in settings where the costs for querying uncertain data directly contribute to the objective value is a promising direction for future work.

# Bibliography


[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*, Prentice Hall, 1993. 14, 42

[ARV08] H. Ackermann, H. Röglin, and B. Vöcking, *On the impact of combinatorial structure on congestion games*, Journal of the ACM **55** (2008), no. 6, 25:1–25:22. 56

[BEY98] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998. 1, 102, 132, 134

[BHKR05] R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman, *Efficient update strategies for geometric computing with uncertainty*, Theory of Computing Systems **38** (2005), 411–423. 4

[BL97] J. R. Birge and F. Louveaux, *Introduction to stochastic programming*, Springer, 1997. 1

[BTEGN09] A. Ben-Tal, L. El Ghaoui, and A. S. Nemirovski, *Robust optimization*, Princeton Series in Applied Mathematics, Princeton University Press, 2009. 1

[CFG+02] M. Charikar, R. Fagin, V. Guruswami, J. M. Kleinberg, P. Raghavan, and A. Sahai, *Query strategies for priced information*, Journal of Computer and System Sciences **64** (2002), no. 4, 785–819. 70

[CH13] G. Charalambous and M. Hoffmann, *Verification problem of maximal points under uncertainty*, Revised Selected Papers of IWOCA, 2013, pp. 94–105. 4

[CK88] R. Chandrasekaran and S. N. Kabadi, *Pseudomatroids*, Discrete Mathematics **71** (1988), no. 3, 205–217. 33

[Dat] `http://www.coga.tu-berlin.de/fileadmin/i26/coga/MSTData.zip`. 48

[DEMM17] C. Dürr, T. Erlebach, N. Megow, and J. Meißner, *Scheduling with explorable uncertainty*, Computing Research Repository (2017). 111

[DEMM18] ———, *Scheduling with explorable uncertainty*, Proceedings of ITCS, 2018. 111

[EH14] T. Erlebach and M. Hoffmann, *Minimum spanning tree verification under uncertainty*, Proceedings of WG, 2014, pp. 164–175. 3, 9, 10, 40, 71, 105, 107

[EH15] ———, *Query-competitive algorithms for computing with uncertainty*, Bulletin of the EATCS **116** (2015). 4, 5, 6, 9, 10, 11, 71

[EHK+08] T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihalák, and R. Raman, *Computing minimum spanning trees with uncertainty*, Proceedings of STACS, 2008, pp. 277–288. 3, 4, 7, 9, 10, 11, 19, 20, 22, 30, 31, 32, 35, 39, 41, 57, 59, 70, 79, 101, 105

[EHK16] T. Erlebach, M. Hoffmann, and F. Kammer, *Query-competitive algorithms for cheapest set problems under uncertainty*, Theoretical Computer Science **613** (2016), 51–64. 3, 4, 9, 10, 23, 32, 33, 55, 56, 59, 62, 63, 65, 82, 84, 152

[FGH+15] S. Fujishige, M. X. Goemans, T. Harks, B. Peis, and R. Zenklusen, *Matroids are immune to braess paradox*, Computing Research Repository (2015). 56

[FMM17] J. Focke, N. Megow, and J. Meißner, *Minimum spanning tree under explorable uncertainty in theory and experiments*, Proceedings of SEA, 2017. 9, 39

[FMO+07] T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy, *Computing shortest paths with uncertainty*, Journal of Algorithms **62** (2007), 1–18. 4

[FMP+03] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom, *Computing the median with uncertainty*, SIAM Journal on Computing **32** (2003), 538–547. 4

[GGI+15] M. Goerigk, M. Gupta, J. Ide, A. Schöbel, and S. Sen, *The robust knapsack problem with queries*, Computers & Operations Research **55** (2015), 12–22. 4, 40, 56, 71, 152

[GSS16] M. Gupta, Y. Sabharwal, and S. Sen, *The update complexity of selection and related problems*, Theory of Computing Systems **59** (2016), no. 1, 112–132. 3, 4, 10, 28, 70, 71, 72, 76, 77, 101, 104

[Guy13] R. Guy, *Unsolved problems in number theory*, vol. 1, Springer Science & Business Media, 2013. 63

[Kah91] S. Kahan, *A model for data in motion*, Proceedings of STOC, 1991, pp. 267–277. 3, 70, 76

[KLS91] B. Korte, L. Lovász, and R. Schrader, *Greedoids*, Springer-Verlag, 1991. 33

[KT01] S. Khanna and W. C. Tan, *On computing functions with uncertainty*, Proceedings of PODS, 2001, pp. 171–182. 4, 69

[KV12] B. Korte and J. Vygen, *Combinatorial optimization*, vol. 21, Springer, 2012. 12, 32

[Lev16] R. Levi, *Practice driven scheduling models*, Talk at Dagstuhl Seminar 16081: Scheduling, 2016. 4, 112

[LMS15] R. Levi, T. Magnanti, and Y. Shaposhnik, *Scheduling with testing*, Manuscript, 2015. 4

[MMS15] N. Megow, J. Meißner, and M. Skutella, *Randomization helps computing a minimum spanning tree under uncertainty*, Proceedings of ESA, 2015. 9, 69

[MMS17] ———, *Randomization helps computing a minimum spanning tree under uncertainty*, SIAM Journal on Computing **46** (2017), no. 4, 1217–1240. 9, 69

[OW00] C. Olston and J. Widom, *Offering a precision-performance tradeoff for aggregation queries over replicated data*, Proceedings of VLDB, 2000, pp. 144–155. 4

[Rei91] G. Reinelt, *TSPLIB – A traveling salesman problem library*, ORSA Journal on Computing **3** (1991), no. 4, 376–384. 40

[Sch02] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*, vol. 24, Springer Science & Business Media, 2002. 59

[Sch05] A. Schrijver, *On the history of combinatorial optimization (till 1960)*, Handbooks in Operations Research and Management Science **12** (2005), 1 – 68. 1

[Sha16] Y. Shaposhnik, *Exploration vs. exploitation: Reducing uncertainty in operational problems*, Ph.D. thesis, Sloan School of Management, MIT, 2016. 4, 112

[Smi56] W. E. Smith, *Various optimizers for single-stage production*, Naval Research Logistics Quarterly **3** (1956), no. 1-2, 59–66. 81

[WW15] Y. Wang and S. C.-W. Wong, *Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm*, Proceedings of ICALP, 2015, pp. 1070–1081. 10, 24, 28

[Yao77] A. C.-C. Yao, *Probabilistic computations: Towards a unified measure of complexity*, Proceedings of FOCS, 1977, pp. 222–227. 102, 132, 134, 149

[Zie12] G. M. Ziegler, *Lectures on polytopes*, vol. 152, Springer Science & Business Media, 2012. 64

# Appendix

# Deterministic Lower Bound

Our lower bound construction considers an instance with uniform upper limit p > 1 and the processing time of every job is either 0 or p. The adversary fixes a fraction $\delta \in [0,1]$ and sets the processing time of a job to p, if and only if the job is tested by the algorithm and among the first $\delta$ fraction of jobs that is either tested or executed untested.

We assume the algorithm knows p and $\delta$, which can only improve the performance of the best-possible deterministic algorithm. The schedule of a deterministic algorithm with best possible competitive ratio has the following form, where $\lambda$, $\nu \geq 0$ and $\nu + \lambda \leq \delta$ : The algorithm first executes $\nu$ n jobs untested, then tests and executes $\lambda$ n long jobs, then tests ($\delta$ - $\nu$ - $\lambda$) n long jobs and delays their execution, then tests and executes the remaining (1 - $\delta$) n short jobs, and finally executes the ($\delta$ - $\nu$ - $\lambda$) n delayed long jobs that were tested earlier. Thus competitive ratio in the limit for n→ ∞ is

```
Clear[p, λ, δ, ν, λsol, δsol, sol];
```

$\rho =$

```
(1 + 2 δ (1 - ν p) + δ^2 (p - 1) + 2 ν (ν + p - 2) + λ^2 + 2 λ (ν + p - 1 - δ p)) / (1 + (p - 1) (δ - ν)^2);
```

Finding the extreme point of the algorithm cost in $\lambda$.

```
λsol = Simplify[Solve[D[ρ, λ] == 0, λ]]
```

$$\{\{\lambda \to 1 + p\ (-1 + \delta) - \nu\}\}$$

Considering the second derivative, yields that this is a minimum of the algorithm cost, as the result is positive

```
Simplify[D[D[ρ, λ], λ]]
```

$$\frac{2}{1 + (-1 + p)\ (\delta - \nu)^2}$$

We check when this minimum is in the feasible region for $\lambda$.

```
λ = λ /. λsol[[1]];
FullSimplify[λ ≤ δ - ν, Assumptions → δ ≤ 1 && p >= 1.5]
```

```
True
```

The condition for $\lambda \geq 0$ is: 1- p (1 - $\delta$) - $\nu \geq 0$

We treat this as a condition on u. Thus, this value for $\lambda$ yields the best algorithm cost, when $\nu$ fulfills the condition above. We now find the minimum when $\nu$ violates the condition. We have seen above, that the first derivative in $\lambda$ is positive, meaning the function increases with $\lambda$. Thus, the optimal choice for the other case is $\lambda$ = 0. This splits the analysis of $\nu$ into the cases $\nu$ in [0, 1 - p(1 - $\delta$)) and $\nu$ in [1 - p(1 - $\delta$), $\delta$].

## Case 1 : $\nu \in [0, 1 - p\ (1 - \delta))$

This case exists if the interval is not empty. This is the case either if $\delta$ = 1 or if 1 - 1/p < $\delta$.

```
Reduce[1 - p (1 - δ) > 0 && 0 ≤ δ ≤ 1, δ]
```

$$(p < 1 \&\& 0 \leq \delta \leq 1) \mid\mid \left(p \geq 1 \&\& \frac{-1 + p}{p} < \delta \leq 1\right)$$

Appendix

```
FullSimplify[ρ]
```

$$\frac{-p^2 \left(-1+\delta\right)^2 + p \left(2+\left(-2+\delta\right) \delta\right) - (\delta-\nu) \left(-2+\delta+\nu\right)}{1+\left(-1+p\right) (\delta-\nu)^2}$$

We consider the change in the algorithm cost as well as the optimal cost, if we decrease $\nu$ by $\epsilon$. Then, the algorithm cost changes by - 2 $\nu$ $\epsilon$ + $\epsilon$^2 + 2 $\epsilon$. The optimal cost changes by (p - 1) (-2 $\nu$ $\epsilon$ + $\epsilon$^2 + 2$\delta$ $\epsilon$), which is positive for any $\epsilon > 0$ independent of $\delta$ and $\nu$ as we have $\nu \leq \delta$. We want to apply Lemma 5.3 to show decreasing $\nu$ also decreases the competitive ratio.Thus we need to show ΔALG/ Δ OPT ≤ ALG/OPT. Here we use, that we already know p ≥ Φ, the golden ratio.

$$\frac{\triangle \text{ALG}}{\triangle \text{OPT}} = \frac{\epsilon \text{^} 2 + 2 \epsilon (1 - \nu)}{(p - 1) (\epsilon \text{^} 2 + 2 \epsilon (\delta - \nu))} \leq \frac{1}{p - 1} < \frac{1}{\Phi} = \Phi < \frac{\text{ALG}}{\text{OPT}}$$

Thus, we can decrease the competitive ratio by decreasing $\nu$. The amount $\epsilon$ by which we can decrease is independent of $\nu$, so we can decrease until $\nu = 0$. Hence, the optimal choice for the algorithm is $\nu = 0$. We call the competitive ratio for this case $\rho 1$.

```
ν = 0;
ρ1 = FullSimplify[ρ]
```

$$\frac{-p^2 \left(-1+\delta\right)^2 - \left(-2+\delta\right) \delta + p \left(2+\left(-2+\delta\right) \delta\right)}{1+\left(-1+p\right) \delta^2}$$

## Case 2 : $\nu \in [1 - p\ (1 - \delta),\ \delta]$

We show the second case always exists, as p must be larger than 1.

```
Reduce[1 - p (1 - δ) ≤ δ && 0 ≤ δ ≤ 1, p]
```

$$p \in \text{Reals} \;\&\&\; \left(\left(0 \leq \delta < 1 \;\&\&\; p \geq 1\right) \;||\; \delta == 1\right)$$

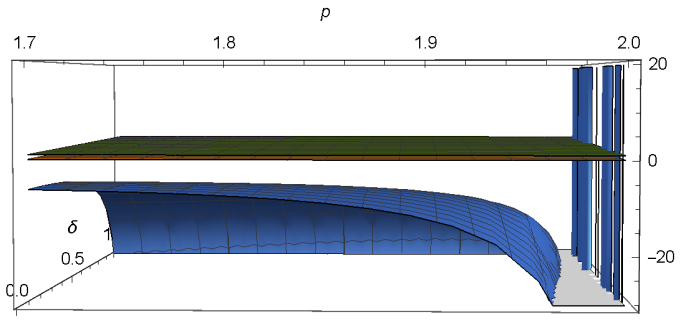
```
Clear[p, δ, ν]; λ = 0;
FullSimplify[ρ]
```

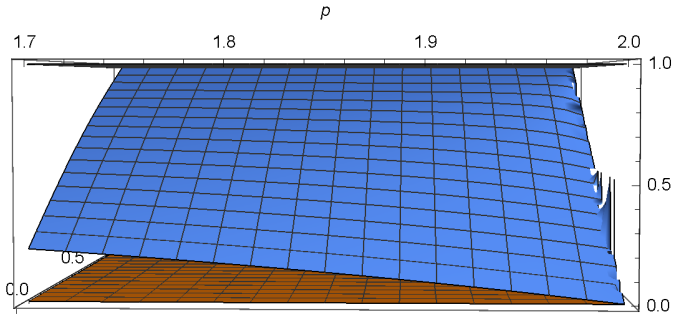
$$\frac{1+\left(-1+p\right) \delta^2 + 2 \nu \left(-2+p+\nu\right) + \delta \left(2-2 p \nu\right)}{1+\left(-1+p\right) (\delta-\nu)^2}$$

```
νsol = Solve[D[ρ, ν] == 0, ν]
```

$$\left\{\left\{\nu \to \left(-6+2p-4\delta+4p\delta+6\delta^2-8p\delta^2+2p^2\delta^2-\sqrt{\left(6-2p+4\delta-4p\delta-6\delta^2+8p\delta^2-2p^2\delta^2\right)^2-4\left(-4+6p-2p^2+4\delta-6p\delta+2p^2\delta\right)\left(-4+2p-2\delta-2p\delta^2+2p^2\delta^2+2\delta^3-2p\delta^3\right)}\right) \Big/ \left(2\left(-4+6p-2p^2+4\delta-6p\delta+2p^2\delta\right)\right)\right\}, \left\{\nu \to \left(-6+2p-4\delta+4p\delta+6\delta^2-8p\delta^2+2p^2\delta^2+\sqrt{\left(6-2p+4\delta-4p\delta-6\delta^2+8p\delta^2-2p^2\delta^2\right)^2-4\left(-4+6p-2p^2+4\delta-6p\delta+2p^2\delta\right)\left(-4+2p-2\delta-2p\delta^2+2p^2\delta^2+2\delta^3-2p\delta^3\right)}\right) \Big/ \left(2\left(-4+6p-2p^2+4\delta-6p\delta+2p^2\delta\right)\right)\right\}\right\}$$

```
v = v /. vsol[[1]];
Plot3D[{0, v, 1}, {p, 1.7, 2}, {δ, 0, 1}, AxesLabel → Automatic]
v = v /. vsol[[2]];
Plot3D[{0, v, 1}, {p, 1.7, 2}, {δ, 0, 1}, AxesLabel → Automatic]
```

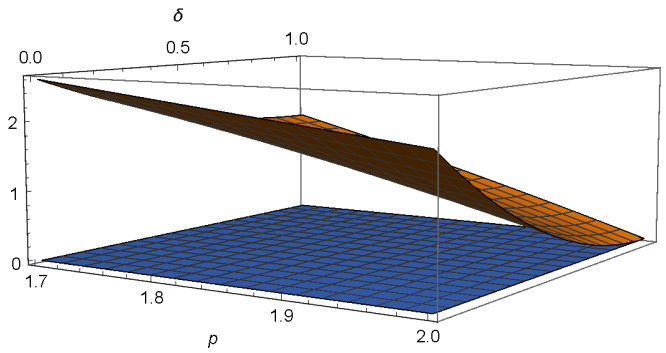



Clearly the first solution is not feasible, as $v$ is negative, so we choose the second solution. We plot the second derivative to see if it is a minimum or maximum.

## Appendix

```
Clear[ν];
sol = FullSimplify[D[D[ρ, ν], ν]];
ν = ν /. νsol[[2]];
Plot3D[{sol, 0}, {p, 1.7, 2}, {δ, 0, 1}, AxesLabel → Automatic]
```




The second derivative in $v$ is positive at the extreme point. Thus, it is a minimum and the algorithm will choose the extreme point for $v$ if it is feasible.

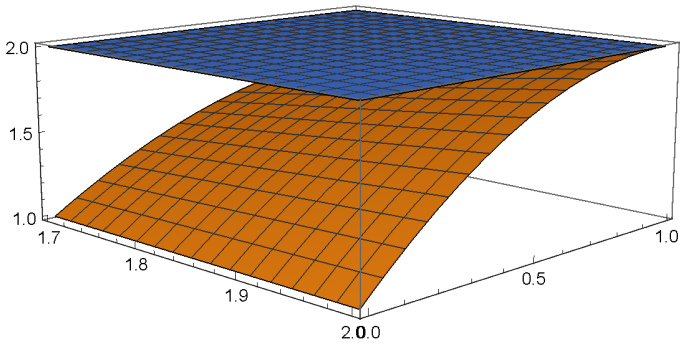
```
Reduce[ν ≤ δ && 0 ≤ δ ≤ 1 && 1 ≤ p ≤ 2]
```

```
False
```

However, the extreme point is never feasible, as we need to ensure $v \leq \delta$. The function is monotonely increasing in $v$ for $v \leq \delta$, as the extreme point is at a larger value of $v$. Thus, the optimal choice for the algorithm is $v = \delta$. For all $\delta \in [0, 1]$ this yields $v \in [1 - p\,(1 - \delta), \delta]$. We call the competitive ratio for this case $\rho 2$.

```
ν = δ; ρ2 = FullSimplify[ρ]
Plot3D[{ρ2, 2}, {p, 1.7, 2}, {δ, 0, 1}]
```

$$1 - (-1 + p)\ (-2 + \delta)\ \delta$$




For $\delta$ < 1 - 1/p there is only one local minimum. This yields the ratio $\rho$2, which is monotonely increasing in $\delta$. Thus, the adversary chooses $\delta \to$ 1 - 1/p.

```
Limit[ρ2, δ → 1 - 1/p]
Maximize[{%, 1 ≤ p ≤ 2}, p]
```

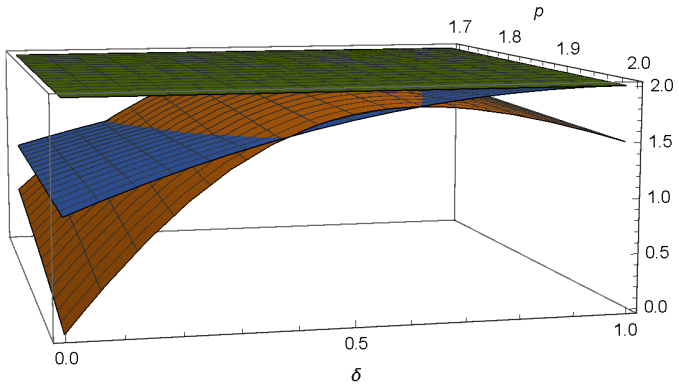
$$\frac{1}{p^2} - \frac{1}{p} + p$$

$$\left\{\frac{7}{4}, \{p \to 2\}\right\}$$

Otherwise, both $\rho$1 and $\rho$2 are local minima and the algorithm chooses the parameter choice attaining the minimum of the two options. Thus, either one ratio dominates the other, or the adversary chooses the range of p and $\delta$ for which they are equal, to maximize the minimum. We show the latter is the case and compute the value p depending on $\delta$ for which the two ratios are equal.

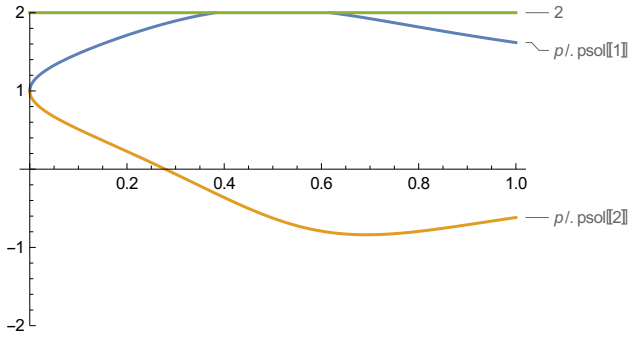
## Appendix

```
Plot3D[{ρ1, ρ2, 2}, {p, 1.7, 2}, {δ, 0, 1}, AxesLabel → Automatic]
psol = FullSimplify[Solve[ρ1 - ρ2 == 0, p]]
```



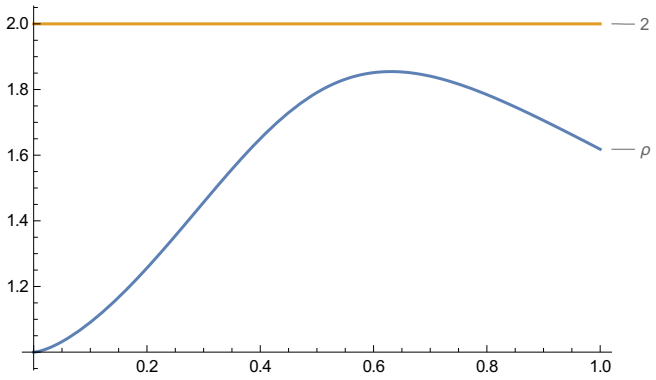

$$\left\{\left\{p \to -\left(\left(2-4\,\delta+\delta^2+4\,\delta^3-2\,\delta^4+\sqrt{\delta}\,\sqrt{8+\left(-2+\delta\right)\,\delta\,\left(10+\delta\,\left(-3+4\,\left(-2+\delta\right)\,\delta\right)\right)}\right)\Big/\left(2\,\left(-1+\left(-2+\delta\right)\,\left(-1+\delta\right)\,\delta\,\left(1+\delta\right)\right)\right)\right)\right\},\ \left\{p \to \left(-2+4\,\delta-\delta^2-4\,\delta^3+2\,\delta^4+\sqrt{\delta}\,\sqrt{8+\left(-2+\delta\right)\,\delta\,\left(10+\delta\,\left(-3+4\,\left(-2+\delta\right)\,\delta\right)\right)}\right)\Big/\left(2\,\left(-1+\left(-2+\delta\right)\,\left(-1+\delta\right)\,\delta\,\left(1+\delta\right)\right)\right)\right\}\right\}$$

```
Plot[{p /. psol[[1]], p /. psol[[2]], 2},
 {δ, 0, 1}, PlotLabels → "Expressions", PlotRange → 2]
```




We see the first solution is the only feasible one. We plot the ratio and compute the value $\delta$, for which it is maximal.

```
p = p /. psol[[1]]; Plot[{ρ, 2}, {δ, 0, 1}, PlotLabels → "Expressions"]
```




```
FullSimplify[ρ]
```

$$\left(-2+\left(-2+\delta\right)\,\delta\,\left(-2+\delta^2+\sqrt{\delta}\,\sqrt{8+\left(-2+\delta\right)\,\delta\,\left(10+\delta\,\left(-3+4\,\left(-2+\delta\right)\,\delta\right)\right)}\,\right)\right)\Big/\left(2\,\left(-1+\left(-2+\delta\right)\,\left(-1+\delta\right)\,\delta\,\left(1+\delta\right)\right)\right)$$

```
δsol = FullSimplify[Solve[D[ρ, δ] == 0]]
N[%]
```

$$\{\{\delta\to 2\},\ \{\delta\to \text{Root}\left[-18+93\,\#1-212\,\#1^2+277\,\#1^3-197\,\#1^4+26\,\#1^5+82\,\#1^6-75\,\#1^7+20\,\#1^8+8\,\#1^9-6\,\#1^{10}+\#1^{11}\,\&,\,1\right]\},\ \{\delta\to \text{Root}\left[-18+93\,\#1-212\,\#1^2+277\,\#1^3-197\,\#1^4+26\,\#1^5+82\,\#1^6-75\,\#1^7+20\,\#1^8+8\,\#1^9-6\,\#1^{10}+\#1^{11}\,\&,\,3\right]\},\ \{\delta\to \text{Root}\left[-18+93\,\#1-212\,\#1^2+277\,\#1^3-197\,\#1^4+26\,\#1^5+82\,\#1^6-75\,\#1^7+20\,\#1^8+8\,\#1^9-6\,\#1^{10}+\#1^{11}\,\&,\,6\right]\},\ \{\delta\to \text{Root}\left[-18+93\,\#1-212\,\#1^2+277\,\#1^3-197\,\#1^4+26\,\#1^5+82\,\#1^6-75\,\#1^7+20\,\#1^8+8\,\#1^9-6\,\#1^{10}+\#1^{11}\,\&,\,7\right]\}\}$$

$$\{\{\delta\to 2.\},\ \{\delta\to -2.06867\},\ \{\delta\to 0.630665\},\ \{\delta\to 0.441156-0.748886\,i\},\ \{\delta\to 0.441156+0.748886\,i\}\}$$

Only the third solution is feasible. We show that for this value, the condition p < 1/(1 - δ) is fulfilled.

```
δ = δ /. δsol[[3]]; Reduce[p < 1 / (1 - δ)]
```

```
True
```

```
N[ρ]
```

```
1.85463
```

# Analysis of RANDOM

Compared to the LaTeX document, the variables T and E are written in lower case, as the keyword E is reserved in Mathematica.

We show in Lemma 5.10 that there is a worst-case instance that consists of four different types of jobs.
(1 - $\alpha$ - $\beta$ - $\gamma$) n jobs j with ub_j = t, p_j = 0 (type 0)
$\alpha$ n jobs j with ub_j = t, p_j = t (type t)
$\beta$ n jobs j with ub_j = e, p_j = e (type e)
$\gamma$ n jobs j with ub_j = e + $\epsilon$, p_j = e + $\epsilon$ (type e+)
for an arbitrary small $\epsilon > 0$, which we will omit in the expressions below.

The algorithm RANDOM tests in a first part all jobs in random order, executes immediately those of size 0, t or e, and defers those of size e+epsilon. In the second part all deferred jobs are executed. The algorithm cost and the optimal cost are

```
Clear[α, β, γ, t, e, p];
ALG = (1 - γ) n (n + 1 + t α n + e β n) / 2 +
  t α n / 2 + e β n / 2 + γ n (n + t α n + e β n) + e γ n (γ n + 1) / 2;
Apart[
 ALG,
 n]
```

$$\frac{1}{2} n \left(1 + t \alpha + e \beta - \gamma + e \gamma\right) + \frac{1}{2} n^2 \left(1 + t \alpha + e \beta + \gamma + t \alpha \gamma + e \beta \gamma + e \gamma^2\right)$$

```
OPT = (1 - α - β - γ) n ((1 - α - β - γ) n + 1) / 2 + (1 - α - β - γ) n (α + β + γ) n +
    t α n (α n + 1) / 2 + t α n (β + γ) n + e β n (β n + 1) / 2 + e β n γ n + e γ n (γ n + 1) / 2 ;
Apart[
 OPT,
 n]
```

$$\frac{1}{2} n \left(1 - \alpha + t \alpha - \beta + e \beta - \gamma + e \gamma\right) +$$
$$\frac{1}{2} n^2 \left(1 - \alpha^2 + t \alpha^2 - 2 \alpha \beta + 2 t \alpha \beta - \beta^2 + e \beta^2 - 2 \alpha \gamma + 2 t \alpha \gamma - 2 \beta \gamma + 2 e \beta \gamma - \gamma^2 + e \gamma^2\right)$$

Our analysis will be in two parts. Both ALG and OPT are expressions of the form n^2 ALG_2 +n ALG_1 and n^2 OPT_2 + n OPT_1. In the first part we analyze the ratio ALG_2/OPT_2 and in the second part the ratio ALG_1/OPT_1.

## Part 1: Quadratic Part of the Ratio

We want to show that ALG_2 ≤ t OPT_2 or in other words t OPT_2 - ALG_2 ≥ 0. Let's denote the left hand side by 'goal'.

Appendix

```
goal =
FullSimplify[t (1 - α^2 - 2 α β - β^2 - 2 α γ - 2 β γ - γ^2 + β^2 e + 2 β γ e + γ^2 e + α^2 t + 2 α β t + 2 α γ t) -
    (1 + γ + β e + β γ e + γ^2 e + α t + α γ t)]
```

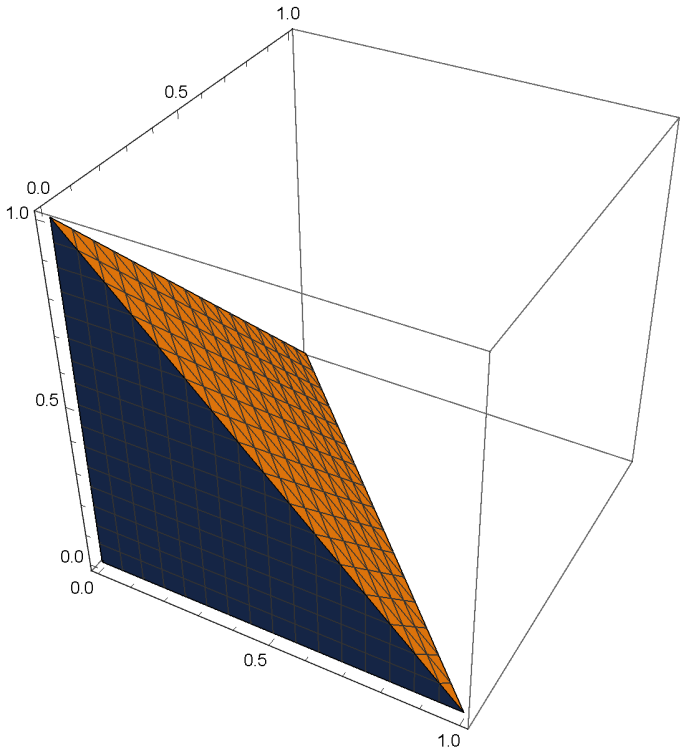
$$-1-\gamma-e\left(\beta+\beta\gamma+\gamma^2\right)+t^2\alpha\left(\alpha+2\left(\beta+\gamma\right)\right)-t\left(-1+\alpha^2-\left(-1+e\right)\left(\beta+\gamma\right)^2+\alpha\left(1+2\beta+3\gamma\right)\right)$$

```
guess = {t → 1.74, e -> 2.86}
```

$$\{t\to 1.74,\ e\to 2.86\}$$

We want to characterize t , e such that for all valid ($\alpha$, $\beta$, $\gamma$) holds G(t, e, $\alpha$, $\beta$, $\gamma$) ≥ 0. These points t, e must satisfy G(t, e, $\alpha$, $\beta$, $\gamma$) ≥ 0 for all local minima ($\alpha$, $\beta$, $\gamma$). Hence we have to find all local minima in the feasible region. Such a point can lie inside the region or on the boundary. In the analysis we will distinguish between the inner region, 2-dimensional open boundaries and 1-dimensional boundaries.

```
RegionPlot3D[α + β + γ ≤ 1, {α, 0, 1}, {β, 0, 1}, {γ, 0, 1}]
```




## Case 3D: Open Polytope

A local minimum ($\alpha$, $\beta$, $\gamma$) which is not on the boundary must in particular be a local minimum in all 3 axis $\alpha$, $\beta$, $\gamma$. We show that with the assumptions we made on T, E the 2nd derivative in $\alpha$ or $\beta$ or $\gamma$ is positive.

```
D[D[goal, α], α]
```

$$-2\,t+2\,t^2$$

```
D[D[goal, β], β]
```

$$2\left(-1+e\right)t$$

```
D[D[goal, γ], γ]
```

$$-2\,e+2\,(-1+e)\,t$$

The second derivative is indeed is positive in all three variables. Hence, we replace each variable with the extreme point for the goal.

```
sol = FullSimplify[Solve[D[goal, α] == 0, α]]
```

$$\left\{\left\{\alpha \to \frac{1-2\,(-1+t)\,\beta+(3-2\,t)\,\gamma}{2\,(-1+t)}\right\}\right\}$$

```
Clear[α]; α = α /. sol[[1]];
```

```
sol = FullSimplify[Solve[D[goal, β] == 0, β]]
```

$$\left\{\left\{\beta \to \frac{1+\gamma-2\,t\,\gamma}{2\,t}\right\}\right\}$$

```
Clear[β]; β = β /. sol[[1]];
```

```
sol = FullSimplify[Solve[D[goal, γ] == 0, γ]]
```

$$\left\{\left\{\gamma \to \frac{(e\,(-1+t)-t)\,(-1+2\,t)}{e\,(-1+t)+t}\right\}\right\}$$

```
Clear[γ]; γ = γ /. sol[[1]];
```

```
FullSimplify[goal]
```

$$\frac{e^2\,(-1+t)^2-e\,t^2+t\,(-1+2\,t)}{e\,(-1+t)+t}$$

We keep only the numerator of the fraction, since the denominator is positive. This yields a first condition.

```
cond1 = FullSimplify[goal (e (t - 1) + t) ≥ 0]
```

$$e^2\,(-1+t)^2+t\,(-1+2\,t) \geq e\,t^2$$

## Case 2D: $\alpha+\beta+\gamma=1$

```
Clear[α, β, γ]; γ = 1 - α - β; FullSimplify[goal]
```

$$-2+\alpha+(-1+t)\,(e\,(-1+\alpha)^2-t\,(-2+\alpha)\,\alpha)+\beta+(-e+t)\,\alpha\,\beta$$

The considered region is defined by $0 \leq \alpha$, $0 \leq \beta$ and $\alpha + \beta \leq 1$.

```
FullSimplify[D[goal, β]]
```

$$1-e\,\alpha+t\,\alpha$$

Since the goal is linear in $\beta$, a local minimum is on the boundary of the region, either for $\beta = 0$ or for $\beta = 1 - \alpha$. These cases are analyzed later.

Appendix

## Case 2D: $\gamma = 0$

```
Clear[α, β, γ]; γ = 0; FullSimplify[goal]
```

$$-1 - e\,\beta + t\left(1 + \alpha\left(-1 + \left(-1 + t\right)\alpha\right) + 2\left(-1 + t\right)\alpha\beta + \left(-1 + e\right)\beta^2\right)$$

We first consider the second derivative in $\alpha$. As it is positive, the extreme point in $\alpha$ is a local minimum and we choose this value for $\alpha$.

```
FullSimplify[D[D[goal, α], α]]
```

$$2\left(-1 + t\right)t$$

```
sol = FullSimplify[Solve[D[goal, α] == 0, α]]
```

$$\left\{\left\{\alpha \to \frac{1}{2\left(-1 + t\right)} - \beta\right\}\right\}$$

```
α = α /. sol[[1]]
```

$$\frac{1}{2\left(-1 + t\right)} - \beta$$

Similarly we proceed for $\beta$.

```
FullSimplify[D[D[goal, β], β]]
```

$$2\left(e - t\right)t$$

```
sol = FullSimplify[Solve[D[goal, β] == 0, β]]
```

$$\left\{\left\{\beta \to \frac{1}{2\,t}\right\}\right\}$$

```
β = β /. sol[[1]]
```

$$\frac{1}{2\,t}$$

If this point is feasible, i.e. fulfills $0 \le \alpha$, $0 \le \beta$, $\alpha + \beta \le 1$, this yield a second condition.

```
FullSimplify[α + β]
FullSimplify[α]
FullSimplify[β]
```

$$\frac{1}{2\left(-1 + t\right)}$$

$$\frac{1}{2\left(-1 + t\right)t}$$

$$\frac{1}{2\,t}$$

```
cond2 = FullSimplify[ goal ≥ 0]
```

$$\frac{1}{1 - t} + 4\,t \ge 4 + \frac{e}{t}$$

We will need this condition later, so we call the lefthandside LHS2.

```
LHS2 = goal;
```

## Case 2D: $\alpha = 0$

```
Clear[α, β, γ]; α = 0; FullSimplify[goal]
```

$$-1-\gamma-e\left(\beta+\beta\gamma+\gamma^2\right)+t\left(1+\left(-1+e\right)\ (\beta+\gamma)^2\right)$$

We first consider the second derivative in $\beta$. As it is positive, the extreme point in $\beta$ is a minimum and we choose this value for $\beta$.

```
FullSimplify[D[D[goal, β], β]]
```

$$2\left(-1+e\right)t$$

```
sol = Solve[D[goal, β] == 0, β]
```

$$\left\{\left\{\beta\to\frac{e+e\gamma+2t\gamma-2et\gamma}{2\left(-1+e\right)t}\right\}\right\}$$

```
β = β /. sol[[1]]
```

$$\frac{e+e\gamma+2t\gamma-2et\gamma}{2\left(-1+e\right)t}$$

The second derivative in $\gamma$ is negative. Hence, the local minimum of this triangle is on the boundary of the triangle.

```
FullSimplify[D[D[goal, γ], γ]]
```

$$\frac{e^2}{2t-2et}$$

## Case 2D: $\beta = 0$

```
Clear[α, β, γ]; β = 0; FullSimplify[goal]
```

$$-1+t^2\alpha\left(\alpha+2\gamma\right)-\gamma\left(1+e\gamma\right)-t\left(-1+\alpha+\alpha^2+3\alpha\gamma+\gamma^2-e\gamma^2\right)$$

We first consider the second derivative in $\alpha$. As it is positive, the extreme point in $\alpha$ is a minimum and we choose this value for $\alpha$.

```
FullSimplify[D[D[goal, α], α]]
```

$$2\left(-1+t\right)t$$

```
sol = Solve[D[goal, α] == 0, α]
```

$$\left\{\left\{\alpha\to\frac{1+3\gamma-2t\gamma}{2\left(-1+t\right)}\right\}\right\}$$

```
α = α /. sol[[1]]
```

$$\frac{1+3\gamma-2t\gamma}{2\left(-1+t\right)}$$

Appendix

```
FullSimplify[D[D[goal, γ], γ]]
```

$$\frac{4\,e\,(-1+t)^2+t\,(-5-4\,(-2+t)\,t)}{2\,(-1+t)}$$

For $\gamma$ we cannot say immediately if the second derivative is positive. We consider the extreme point.

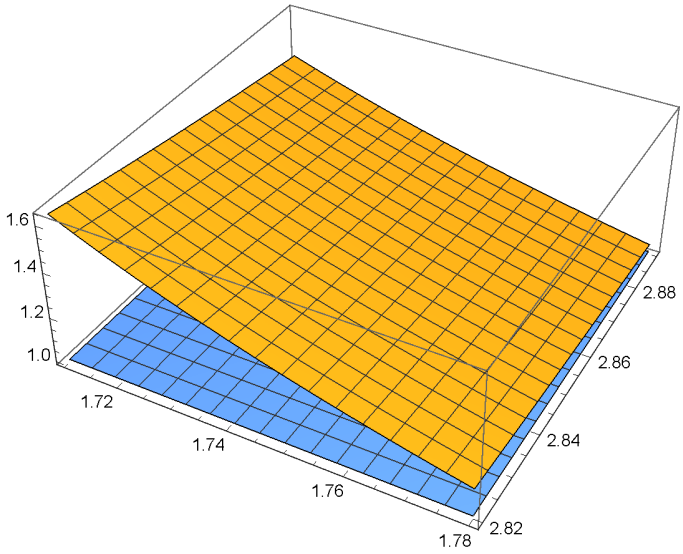
```
sol = FullSimplify[Solve[D[goal, γ] == 0, γ]]
```

$$\left\{\left\{\gamma \to \frac{(-2+t)\,(-1+2\,t)}{-4\,e\,(-1+t)^2+t\,(5+4\,(-2+t)\,t)}\right\}\right\}$$

```
γ = γ /. sol[[1]]
```

$$\frac{(-2+t)\,(-1+2\,t)}{-4\,e\,(-1+t)^2+t\,(5+4\,(-2+t)\,t)}$$

```
Plot3D[{α + γ, 1}, {t, 1.71, 1.78}, {e, 2.82, 2.89}]
```




The considered point is outside the triangle for the range (T,E) we are interested in, thus it does not matter if it is a minimum or a maximum in $\gamma$.

## Case 1D: ($\alpha$, $\beta$, $\gamma$) = (x, 1 - x, 0)

```
Clear[α, β, γ]; α = x; β = 1 - α; γ = 0; FullSimplify[goal]
```

$$-1+e\,(1+t\,(-1+x))\,(-1+x)-t\,(1+t\,(-2+x))\,x$$

We consider the second derivative in x. As it is positive, the extreme point in x is a minimum and we choose this value for x.

```
FullSimplify[D[D[goal, x], x]]
```

$$2\,(e-t)\,t$$

```
sol = Solve[D[goal, x] == 0, x]
```

$$\left\{\left\{x \to \frac{-1+2\,t}{2\,t}\right\}\right\}$$

So x is between 0 and 1 as we desired.

```
x = x /. sol[[1]]
```

$$\frac{-1+2\,t}{2\,t}$$

```
cond3 = FullSimplify[goal] ≥ 0
```

$$-\frac{3}{4}-\frac{e}{4\,t}+(-1+t)\,t \geq 0$$

## Case 1D: (α, β, γ) = (x, 0, 1 - x)

```
Clear[α, β, γ, x]; α = x; β = 0; γ = 1 - x; FullSimplify[goal]
```

$$-2+x+(-1+t)\,\left(e\,(-1+x)^2-t\,(-2+x)\,x\right)$$

We consider the second derivative in x. As it is positive, the extreme point in x is a minimum and we choose this value for x.

```
FullSimplify[D[D[goal, x], x]]
```

$$2\,(e-t)\,(-1+t)$$

```
sol = Solve[D[goal, x] == 0, x]
```

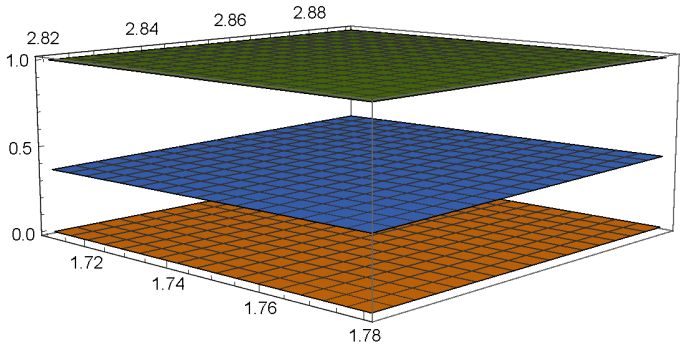
$$\left\{\left\{x \to \frac{-1-2\,e+2\,t+2\,e\,t-2\,t^2}{2\,(e-t)\,(-1+t)}\right\}\right\}$$

```
x = x /. sol[[1]]
```

$$\frac{-1-2\,e+2\,t+2\,e\,t-2\,t^2}{2\,(e-t)\,(-1+t)}$$

Appendix

```
Plot3D[{0, x, 1}, {t, 1.71, 1.78}, {e, 2.82, 2.89}]
```




This is good, be cause x is in [0, 1] as we desired.

```
cond4 = FullSimplify[goal 4 (e - t) (t - 1)] ≥ 0
```

$$-\left(1-2\left(-1+t\right)t\right)^2+4e\left(1+\left(-2+t\right)t^2\right)\geq 0$$

We will need this condition later, so we call the lefthand side LHS4.

```
LHS4 = goal;
```

## Case 1D: ($\alpha$, $\beta$, $\gamma$)=(0, x, 1 - x)

```
Clear[α, β, γ, x]; α = 0; β = x; γ = 1 - x; FullSimplify[goal]
```

$$-2+e\left(-1+t\right)+x$$

The local minimum is at x=0.

```
x = 0;
```

```
cond5 = FullSimplify[goal ≥ 0]
```

$$et\geq 2+e$$

## Case 1D: ($\alpha$, 0, 0)

```
Clear[α, β, γ, x]; β = 0; γ = 0; FullSimplify[goal]
```

$$-1+t-t\alpha+\left(-1+t\right)t\alpha^2$$

We compute the second derivative in $\alpha$ and see that it is positive.

```
FullSimplify[D[D[goal, α], α]]
```

$$2\,(-1+t)\,t$$

```
sol = FullSimplify[Solve[D[goal, α] == 0, α]]
```

$$\left\{\left\{\alpha \to \frac{1}{2\,(-1+t)}\right\}\right\}$$

The extreme point in $\alpha$ is in [0,1] and thus feasible.

```
α = α /. sol[[1]]
```

$$\frac{1}{2\,(-1+t)}$$

```
FullSimplify[goal]
```

$$-\frac{5}{4}+\frac{1}{4-4\,t}+t$$

```
cond6 = FullSimplify[4 goal ≥ 0]
```

$$\frac{1}{1-t}+4\,t \geq 5$$

## Case 1D: (0, $\beta$, 0)

```
Clear[α, β, γ, x]; α = 0; γ = 0; FullSimplify[goal]
```

$$-1+t-e\,\beta+(-1+e)\,t\,\beta^2$$

We compute the second derivative in $\beta$ and see that it is positive.

```
FullSimplify[D[D[goal, β], β]]
```

$$2\,(-1+e)\,t$$

```
sol = Solve[D[goal, β] == 0, β]
```

$$\left\{\left\{\beta \to \frac{e}{2\,(-1+e)\,t}\right\}\right\}$$

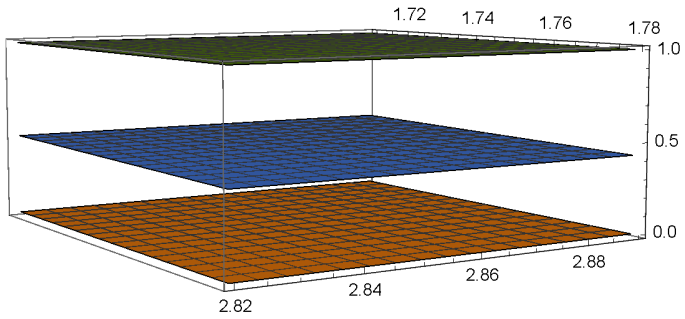
```
β = β /. sol[[1]]
```

$$\frac{e}{2\,(-1+e)\,t}$$

We ensure that the value for $\beta$ is feasible.

Appendix

```
Plot3D[{0, β, 1}, {t, 1.71, 1.78}, {e, 2.82, 2.89}]
```




```
cond7 = FullSimplify[goal ≥ 0]
```

$$4 + \frac{e^2}{(-1+e)\,t} \le 4\,t$$

## Case 1D: (0, 0, γ)

```
Clear[α, β, γ, x]; α = 0; β = 0; FullSimplify[goal]
```

$$-1 + t + (-1+e)\,t\,\gamma^2 - \gamma\,(1+e\,\gamma)$$

We compute the second derivative in γ and see that it is positive.

```
D[D[goal, γ], γ]
```

$$-2\,e + 2\,(-1+e)\,t$$

```
sol = Solve[D[goal, γ] == 0, γ]
```

$$\left\{\left\{\gamma \to \frac{1}{2\,(-e-t+e\,t)}\right\}\right\}$$

```
γ = γ /. sol[[1]]
```

$$\frac{1}{2\,(-e-t+e\,t)}$$

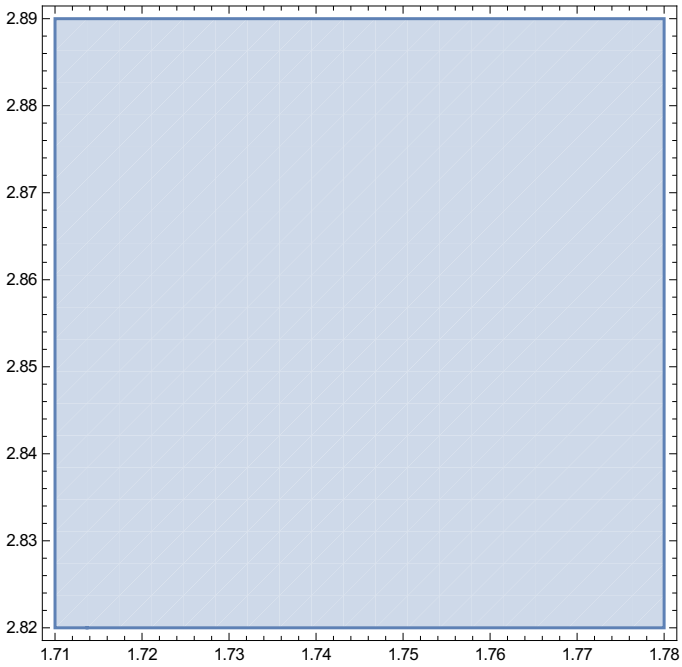
```
cond8 = FullSimplify[goal] ≥ 0
```

$$-1 + t + \frac{1}{4\,e + 4\,t - 4\,e\,t} \ge 0$$

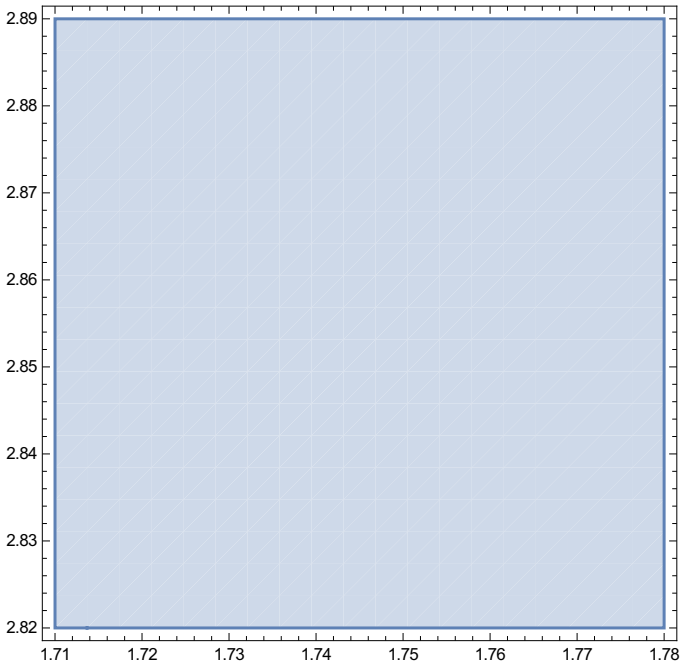
# Finding optimal T, E satisfying all conditions

We first show that conditions 3, 5, 6, and 7 are fulfilled for all values of (T, E) we consider.

```
Clear[α, β, γ, x]; RegionPlot[{cond3}, {t, 1.71, 1.78}, {e, 2.82, 2.89}]
```

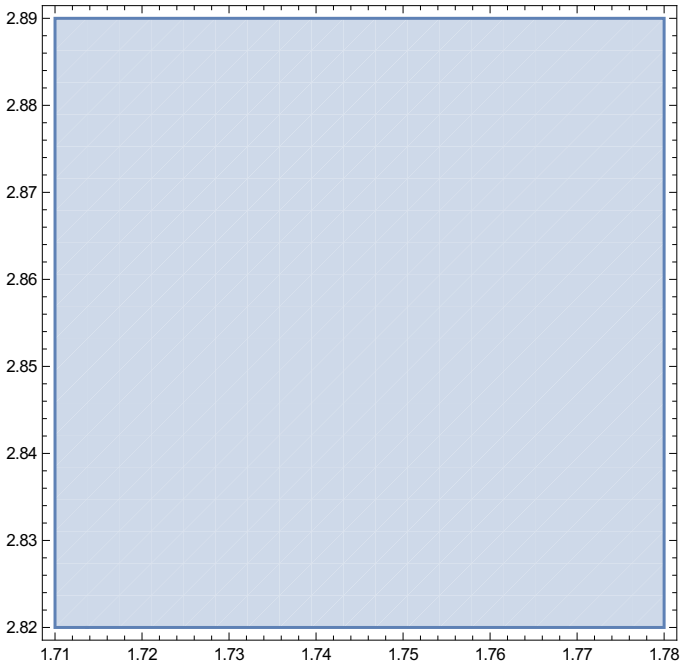



```
RegionPlot[{cond5}, {t, 1.71, 1.78}, {e, 2.82, 2.89}]
```

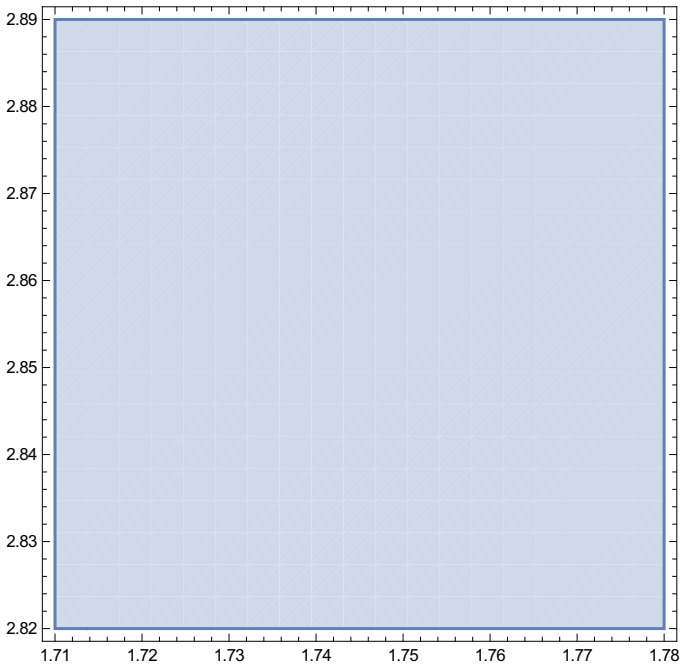


## Appendix

```
RegionPlot[{cond6}, {t, 1.71, 1.78}, {e, 2.82, 2.89}]
```

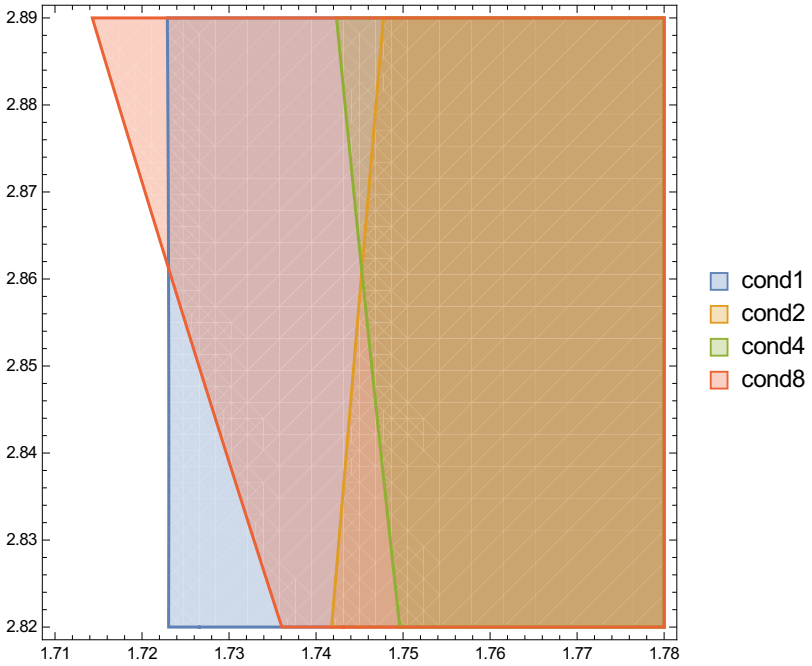



```
RegionPlot[{cond7}, {t, 1.71, 1.78}, {e, 2.82, 2.89}]
```




For the other conditions, we show that condition 2 and 4 are most restrictive.

```
RegionPlot[{cond1, cond2, cond4, cond8},
 {t, 1.71, 1.78}, {e, 2.82, 2.89}, PlotLegends → "Expressions"]
```




The point which fulfills both conditions 2 and 4 and has smallest T is the point where both conditions are tight.

**cond2**

**cond4**

$$\frac{1}{1-t}+4\,t\geq 4+\frac{e}{t}$$

$$-\left(1-2\left(-1+t\right)t\right)^2+4\,e\left(1+\left(-2+t\right)t^2\right)\geq 0$$

We set condition 2 equal to zero to find the dependence between T and E.

```
sole = Solve[LHS2 == 0, e]
```

$$\left\{\left\{e\to\frac{t\left(3-8\,t+4\,t^2\right)}{-1+t}\right\}\right\}$$

We set E to this value. Additionally condition 4 has to be tight, so we solve for T to find the best choice.

```
e = e /. sole[[1]]; solt = FullSimplify[Solve[LHS4 == 0, t]]
```

$$\left\{\left\{t\to\text{Root}\left[-1-16\,\#1+20\,\#1^2+36\,\#1^3-52\,\#1^4+16\,\#1^5\,\&,\,1\right]\right\},\right.$$
$$\left\{t\to\text{Root}\left[-1-16\,\#1+20\,\#1^2+36\,\#1^3-52\,\#1^4+16\,\#1^5\,\&,\,2\right]\right\},$$
$$\left\{t\to\text{Root}\left[-1-16\,\#1+20\,\#1^2+36\,\#1^3-52\,\#1^4+16\,\#1^5\,\&,\,3\right]\right\},$$
$$\left\{t\to\text{Root}\left[-1-16\,\#1+20\,\#1^2+36\,\#1^3-52\,\#1^4+16\,\#1^5\,\&,\,4\right]\right\},$$
$$\left.\left\{t\to\text{Root}\left[-1-16\,\#1+20\,\#1^2+36\,\#1^3-52\,\#1^4+16\,\#1^5\,\&,\,5\right]\right\}\right\}$$

Appendix

```
N[solt]
```

$\{\{t \to -0.600748\}, \{t \to -0.0586886\}, \{t \to 0.688139\}, \{t \to 1.47604\}, \{t \to 1.74526\}\}$

Only one of the solutions is in the interval [1.5, 2], so we choose this one.

```
t = t /. solt[[5]]; N[t]
N[e]
```

```
1.74526
```

```
2.86091
```

# Part 2: Linear Part of the Ratio

```
Clear[α, β, γ, t, e, p, goal];
goal = FullSimplify[t (1 - α - β - γ + β e + γ e + α t) - (1 - γ + β e + γ e + α t)]
```

$-1 + t^2 \alpha + \gamma - e (\beta + \gamma) - t (-1 + 2 \alpha + \beta - e \beta + \gamma - e \gamma)$

We consider the first derivative to find the extreme point in $\alpha$, $\beta$, and $\gamma$.

```
D[goal, α]
N[% /. sole /. solt[[5]]]
```

$-2 t + t^2$

$\{-0.444583\}$

```
D[goal, β]
N[% /. sole /. solt[[5]]]
```

$-e - (1 - e) t$

$\{0.386867\}$

```
D[goal, γ]
N[% /. sole /. solt[[5]]]
```

$1 - e - (1 - e) t$

$\{1.38687\}$

The first derivative in $\alpha$ is negative and for the other two variables it is positive, so the worst case is $\alpha = 1$ and $\beta = \gamma = 0$.

```
goal /. {α → 1, β → 0, γ → 0}
```

$-1 - t + t^2$

```
N[% /. sole /. solt[[5]]]
```

```
{0.30068}
```

This means that for the chosen values of T, E we have ALG_1/OPT_1 ≤ T. Since we also have ALG_2/OPT_2 ≤ T, the expression (n^2 ALG_2 + n ALG_1 )/(n^2 OPT_2 + n OPT_1) is at most T as well.

# Analysis of BEAT for 1.5<p<2

```
Clear[α, p, δ, δ2, ratio, R1, R2, αsol, δsol];
```

BEAT is an algorithm for instances with uniform upper limit p. We show its competitive ratio is bounded by the following function

$$\frac{1+2\,(-2+p)\,p+\sqrt{(1-2\,p)^2\,(-3+4\,p)}}{2\,(-1+p)\,p};$$

With BEAT, we aim to balance the time testing jobs and the time executing jobs while there are untested jobs. A job is called *short* if its running time is at most E = max {1, p - 1} and *long* otherwise. We iterate testing an arbitrary job and then execute the job with smallest processing time either, if it is a short job, or if the difference between the total time that long jobs have been tested and the total time long jobs have been executed exceeds the job's processing time. Once all jobs have been tested, we execute the remaining jobs in order of non - decreasing processing time.
We are in the case that the uniform upper limit p is less than 2, which means all jobs with processing time larger than 1 are long jobs (E=1 in the notation of the paper). Let $\alpha > 0$ be the ratio between long and short jobs and let $0 \le \delta \le 1$ be the fraction of short jobs with processing time 1. We show in Lemma 9 and 10 that the asymptotic competitive ratio is

```
ratio = ((p + 2 - 1/p) + α^2 (2 (2 δ - δ^2) + (1 - δ)^2) + 2 α (2 + (1 - 1/p) (1 + δ)))/
    (p + α^2 ((p - 1) δ^2 + 1) + 2 α (1 + (p - 1) δ));
FullSimplify[
 ratio]
```

$$\frac{-1+p^2-2\,\alpha\,(1+\delta)+p\,(2+\alpha\,(6+\alpha+2\,(1+\alpha)\,\delta-\alpha\,\delta^2))}{p\,(p\,(1+\alpha\,\delta)^2-\alpha\,(-1+\delta)\,(2+\alpha+\alpha\,\delta))}$$

The function ratio is maximal for $\delta = 0$, $\delta = 1$ or where the first derivative is 0, so we distinguish these three cases.

## Case: $\delta$=0

```
δ = 0; Collect[ratio , α]
```

$$\frac{2-\frac{1}{p}+p+2\left(3-\frac{1}{p}\right)\alpha+\alpha^2}{p+2\,\alpha+\alpha^2}$$

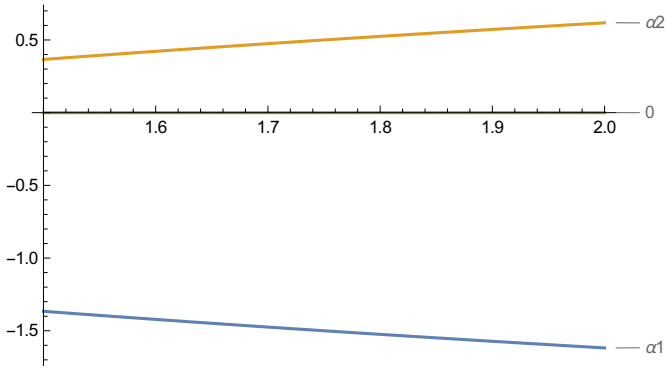
We find the extreme point in $\alpha$ of this function and show the second derivative is negative. This means we have a maximum and we do not need to consider the two interval bounds for $\alpha$.

```
αsol = FullSimplify[Solve[D[ratio, α] == 0, α]]
```

$$\left\{\left\{\alpha\to\frac{-1+2\,p+\sqrt{(-1+2\,p)^2\,(-3+4\,p)}}{2-4\,p}\right\},\ \left\{\alpha\to\frac{1-2\,p+\sqrt{(-1+2\,p)^2\,(-3+4\,p)}}{-2+4\,p}\right\}\right\}$$

We plot the two solutions to see which is feasible.

```
Clear[α1]; α1 = α /. αsol[[1]]; α2 = α /. αsol[[2]];
Plot[{α1, α2, 0}, {p, 1.5, 2}, PlotLabels → "Expressions"]
```

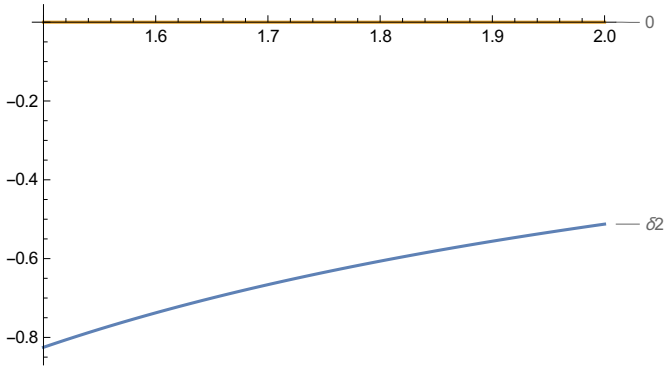



Clearly, $\alpha 1$ is not feasible as it is negative. Thus, the ratio for $\delta = 0$ has a single maximum or minimum in the feasible range for $\alpha$.

```
Clear[α]; δ2 = FullSimplify[D[D[ratio, α], α]]
```

$$-\frac{2\,(-1+2\,p)\,\left(-4+p\,(5+6\,\alpha)-\alpha\,(6+\alpha\,(3+2\,\alpha))\right)}{p\,(p+\alpha\,(2+\alpha))^3}$$

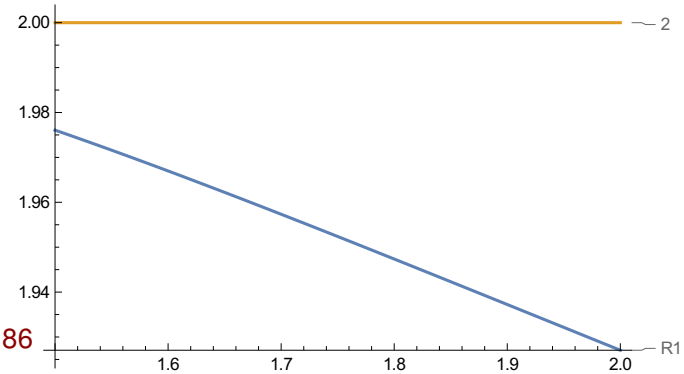
```
α = α2; Plot[{δ2, 0}, {p, 1.5, 2}, PlotLabels → "Expressions"]
```




The second derivative is negative and thus the extreme point of a is a maximum, as we claimed. This yields a lower bound on the ratio, which we call R1. Out of interest, we plot it.

```
R1 = FullSimplify[ratio]
Plot[{R1, 2}, {p, 1.5, 2}, PlotLabels → "Expressions"]
```

$$\frac{1+2\,(-2+p)\,p+\sqrt{(1-2\,p)^2\,(-3+4\,p)}}{2\,(-1+p)\,p}$$



## Case: $\delta = 1$

```
Clear[δ, α]; δ = 1; Collect[ratio, α]
```

$$\frac{2 - \frac{1}{p} + p + 2\left(2 + 2\left(1 - \frac{1}{p}\right)\right)\alpha + 2\alpha^2}{p + 2p\alpha + p\alpha^2}$$

We consider the first derivative in $\alpha$ to show the function is monotonically decreasing for increasing $\alpha$.

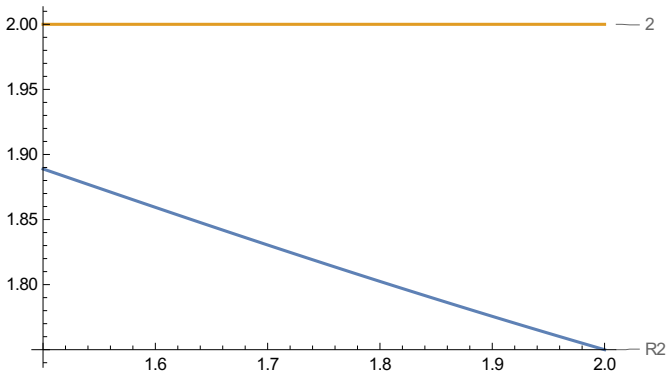
```
FullSimplify[D[ratio, α]]
```

$$-\frac{2(-1+p)(-1+p+2\alpha)}{p^2(1+\alpha)^3}$$

As we have p > 1 and $\alpha > 0$, both the numerator and the denominator of the function are positive. Hence, the first derivative is negative for all feasible values of $\alpha$ and p. This means the function is monotonically decreasing for increasing $\alpha$. It's maximal value thus is attained for $\alpha = 0$. This yields a second lower bound on the ratio, which we call R2.

```
α = 0; R2 = FullSimplify[ratio]
Plot[{R2, 2}, {p, 1.5, 2}, PlotLabels → "Expressions"]
```

$$\frac{-1 + p(2 + p)}{p^2}$$




## Case: $\delta$ at the Extreme Point

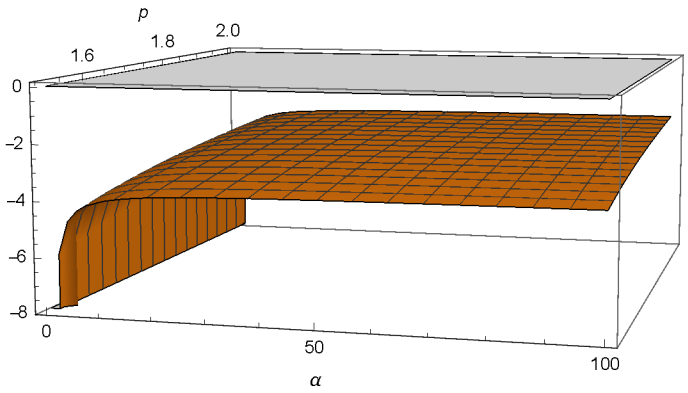
As the third case we consider the value of $\delta$, for which the first derivative in $\delta$ of the ratio function is 0.

```
Clear[α, δ]; δsol = FullSimplify[Solve[D[ratio, δ] == 0, δ]]
```

$$\{\{\delta \to (-\alpha(1 + 2\alpha + p(-3 - 6\alpha + p(2 + p + \alpha(6 + \alpha))))) + \sqrt{(\alpha^2(-3(1 + 4\alpha) + p(14 + p^5 + 2p^4(-2 + \alpha(4 + \alpha))) + 4\alpha(17 + \alpha(5 + 2\alpha)) + p^3(10 + \alpha(-24 + \alpha(2 + \alpha)(6 + \alpha))) + 2p^2(1 + \alpha(46 + \alpha(13 + 2\alpha(4 + \alpha)))) - p(19 + 2\alpha(64 + \alpha(27 + 2\alpha(7 + \alpha))))))))} / (2(-1 + p)\alpha^2(-1 + p(2 + \alpha)))\},$$
$$\{\delta \to -((\alpha(1 + 2\alpha + p(-3 - 6\alpha + p(2 + p + \alpha(6 + \alpha))))) + \sqrt{(\alpha^2(-3(1 + 4\alpha) + p(14 + p^5 + 2p^4(-2 + \alpha(4 + \alpha))) + 4\alpha(17 + \alpha(5 + 2\alpha)) + p^3(10 + \alpha(-24 + \alpha(2 + \alpha)(6 + \alpha))) + 2p^2(1 + \alpha(46 + \alpha(13 + 2\alpha(4 + \alpha)))) - p(19 + 2\alpha(64 + \alpha(27 + 2\alpha(7 + \alpha))))))))} / (2(-1 + p)\alpha^2(-1 + p(2 + \alpha)))\}\}$$

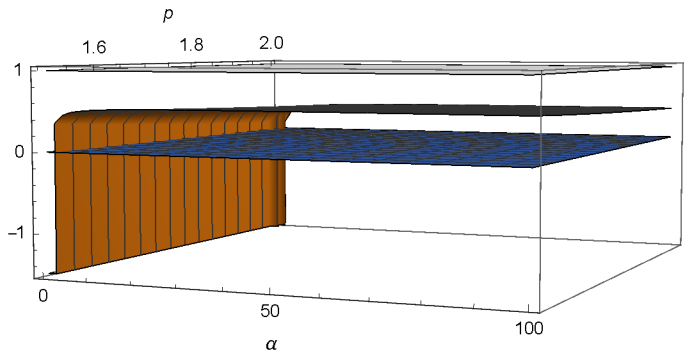
We first show the second solution is not in the feasible interval for $\delta$.

```
Clear[δ]; δ = δ /. δsol[[2]] ;
Plot3D[{δ, 0}, {α, 0, 100}, {p, 1.5, 2}, AxesLabel → Automatic]
```



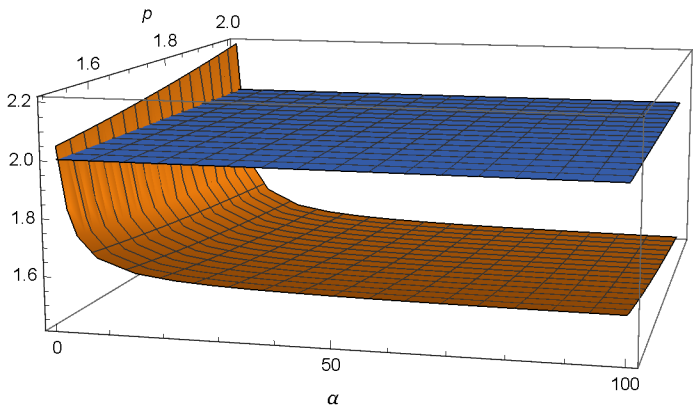

We check if $\delta$ is feasible for the first solution (we need $0 \le \delta \le 1$).

```
Clear[δ]; δ = δ /. δsol[[1]] ;
Plot3D[{δ, 0, 1}, {α, 0, 100}, {p, 1.5, 2}, AxesLabel → Automatic]
```




We see $\delta$ is feasible if the variable $\alpha$ is not too small. Otherwise, there is no extreme point of ratio in the feasible interval for $\delta$, which means one of the previous two cases $\delta = 0$ and $\delta = 1$ discovered the maximum. Consider the ratio.

```
Plot3D[{ratio, 2}, {α, 0, 100}, {p, 1.5, 2}, AxesLabel → Automatic]
```

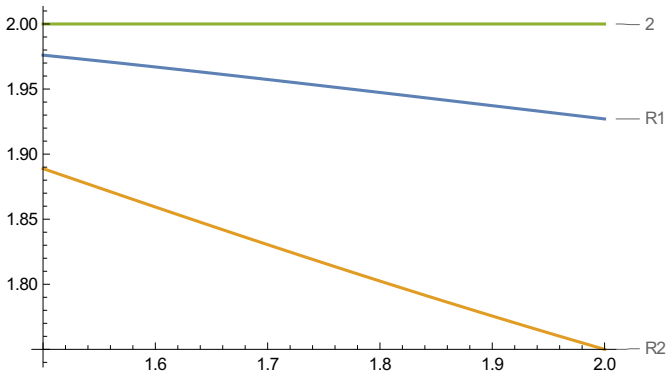



We observe the competitive ratio increases when a decreases, independent of the upper limit p. Thus, the adversary chooses the smallest feasible value for $\alpha$. However, for this value, we have $\delta$ = 0, which is the case we have already considered above. Thus we do not get a new bound on the competitive ratio for this case.

## Summary

We found two bounds on the competitive ratio: R1 and R2. We plot them to show R1 is always larger in the interval for p which we consider. Thus R1 is the bound we obtain.

```
Plot[{R1, R2, 2}, {p, 1.5, 2}, PlotLabels → Automatic]
```



# Analysis of BEAT for 2 ≤ p < 3

```
Clear[α, p, δ, αsol, δsol, R1, R2];
```

BEAT is an algorithm for instances with uniform upper limit p. We show its competitive ratio is bounded by the following function

$$\frac{1+2\,(-2+p)\,p+\sqrt{(1-2\,p)^2\,(-3+4\,p)}}{2\,(-1+p)\,p};$$

With BEAT, we aim to balance the time testing jobs and the time executing jobs while there are untested jobs. A job is called *short* if its running time is at most E = max {1, p - 1} and *long* otherwise. We iterate testing an arbitrary job and then execute the job with smallest processing time either, if it is a short job, or if the difference between the total time that long jobs have been tested and the total time long jobs have been executed exceeds the job's processing time. Once all jobs have been tested, we execute the remaining jobs in order of non - decreasing processing time.
We are in the case that the uniform upper limit p is at least 2, which means all jobs with processing time larger than p -1 are long jobs (E = p - 1 in the notation of the paper). Let $\alpha > 0$ be the ratio between long and short jobs and let $0 \le \delta \le 1$ be the fraction of short jobs with processing time p - 1. We show in Lemma 9 and 10 that the asymptotic competitive ratio is

```
ratio = ((p + 2 - 1/p) + α^2 (p (2 δ - δ^2) + (1 - δ)^2) + 2 α (2 + (1 - 1/p) (1 + (p - 1) δ)))/
    (p + α^2 ((p - 1) δ^2 + 1) + 2 α (1 + (p - 1) δ));
FullSimplify[
 ratio]
```

$$\left(-1-2\,\alpha+p\,\left(2+p+\alpha\,\left(6+\alpha\right)\right)+2\,\left(-1+p\right)\,\alpha\,\left(-1+p+p\,\alpha\right)\,\delta-\left(-1+p\right)\,p\,\alpha^2\,\delta^2\right)/\left(p\,\left(p\,\left(1+\alpha\,\delta\right)^2-\alpha\,\left(-1+\delta\right)\,\left(2+\alpha+\alpha\,\delta\right)\right)\right)$$

The function ratio is maximal for $\delta = 0$, $\delta = 1$ or where the first derivative is 0, so we distinguish these three cases.

## Case : $\delta = 0$

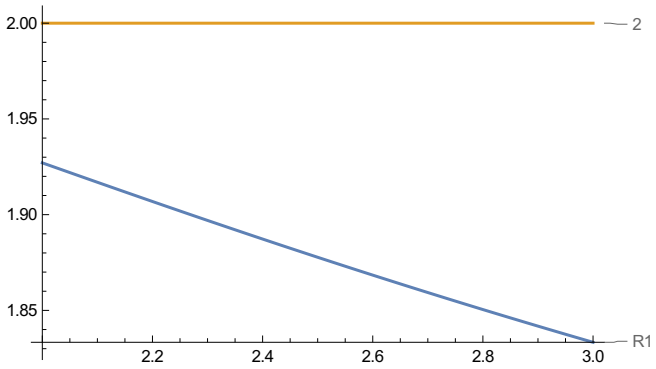
```
δ = 0; Collect[ratio , α]
```

$$\frac{2-\frac{1}{p}+p+2\,\left(3-\frac{1}{p}\right)\,\alpha+\alpha^2}{p+2\,\alpha+\alpha^2}$$

This is exactly the same expression we had for small p. Thus, this case also yields the same lower bound R1 on the asymptotic competitive ratio of the algorithm.

Appendix

```
R1 = (1 + 2 (-2 + p) p + Sqrt[(1 - 2 p)^2 (-3 + 4 p)]) / (2 (-1 + p) p);
Plot[{R1, 2}, {p, 2, 3}, PlotLabels → "Expressions"]
```




## Case: $\delta = 1$

```
Clear[δ, α]; δ = 1; Collect[ratio, α]
```

$$\frac{2 - \frac{1}{p} + p + 2\left(2 + \left(1 - \frac{1}{p}\right) p\right)\alpha + p\,\alpha^2}{p + 2\,p\,\alpha + p\,\alpha^2}$$

We consider the first derivative in $\alpha$, to show the function is monotonically decreasing for increasing $\alpha$.

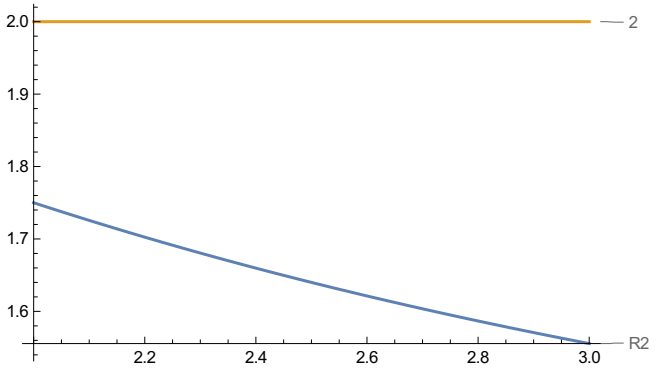
```
FullSimplify[D[ratio, α]]
```

$$-\frac{2\left(-1 + p + p\,\alpha\right)}{p^2\left(1 + \alpha\right)^3}$$

As we have p > 1 and $\alpha > 0$, both the numerator and the denominator of the function are positive. Hence, the first derivative is negative for all feasible values of $\alpha$ and p. This means the function is monotonically decreasing for increasing $\alpha$. It's maximal value thus is attained for $\alpha = 0$. This yields a second lower bound on the ratio, which we call R2.

```
α = 0; R2 = FullSimplify[ratio]
Plot[{R2, 2}, {p, 2, 3}, PlotLabels → "Expressions"]
```

$$\frac{-1 + p\,(2 + p)}{p^2}$$



## Case: $\delta$ at the ExtremePpoint

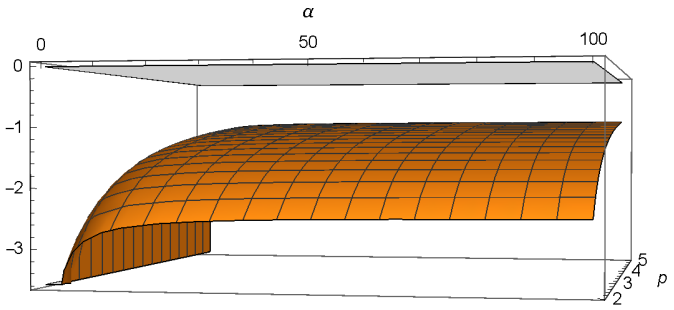
As the third case we consider the value of $\delta$, for which the first derivative in $\delta$ of the ratio function is 0.

```
Clear[α, δ]; δsol = FullSimplify[Solve[D[ratio, δ] == 0, δ]]
```

$$\{\{\delta \to \left(-\left(-1+p\right)\alpha\left(-1-2\alpha+2p\left(1+p+\alpha\left(4+\alpha\right)\right)\right)+\sqrt{\left(-1+p\right)^2\alpha^2\left(5+4\alpha+4p\left(1+\alpha\right)\left(-7-4\alpha+p\left(11+p^2\left(1+\alpha\right)+2\alpha\left(6+\alpha\right)+p\left(1+\alpha\right)\left(-4+\alpha\left(2+\alpha\right)\right)\right)\right)\right)}\right)/\left(2\left(-1+p\right)^2\alpha^2\left(-1+p\left(2+\alpha\right)\right)\right)\},\ \{\delta \to -\left(\left(\left(-1+p\right)\alpha\left(-1-2\alpha+2p\left(1+p+\alpha\left(4+\alpha\right)\right)\right)+\sqrt{\left(-1+p\right)^2\alpha^2\left(5+4\alpha+4p\left(1+\alpha\right)\left(-7-4\alpha+p\left(11+p^2\left(1+\alpha\right)+2\alpha\left(6+\alpha\right)+p\left(1+\alpha\right)\left(-4+\alpha\left(2+\alpha\right)\right)\right)\right)\right)}\right)/\left(2\left(-1+p\right)^2\alpha^2\left(-1+p\left(2+\alpha\right)\right)\right)\right)\}\}$$

We first show the second solution is not in the feasible interval for $\delta$.

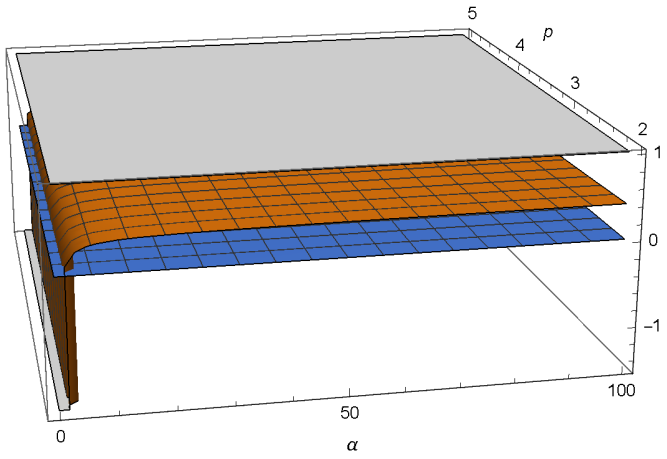
```
Clear[δ]; δ = δ /. δsol[[2]] ;
Plot3D[{δ, 0}, {α, 0, 100}, {p, 2, 5}, AxesLabel → Automatic]
```




We check if $\delta$ is feasible for the first solution (we need $0 \le \delta \le 1$).

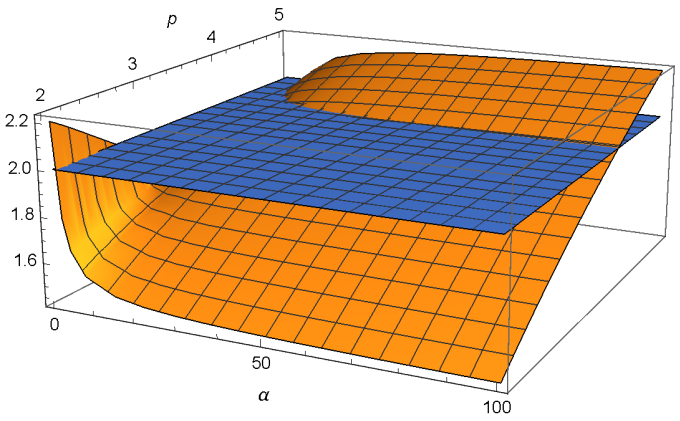
## Appendix

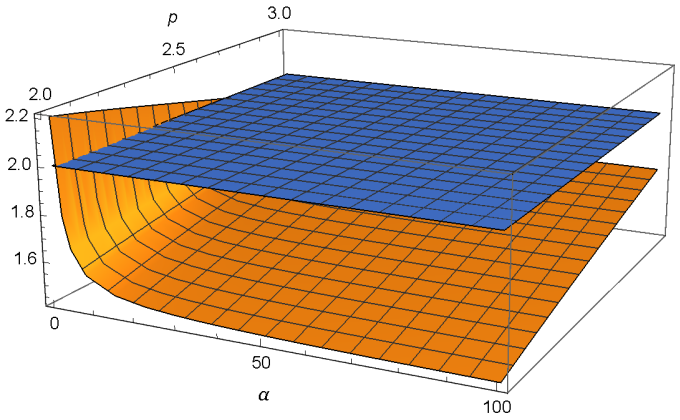
```
Clear[δ]; δ = δ /. δsol[[1]] ;
Plot3D[{δ, 0, 1}, {α, 0, 100}, {p, 2, 5}, AxesLabel → Automatic]
```




We see the first solution is feasible if the variable $\alpha$ is not too small. Otherwise, there is no extreme point of ratio in the feasible interval for $\delta$, which means one of the previous two cases $\delta = 0$ and $\delta = 1$ discovered the maximum. Let us consider the competitive ratio for $\delta$ at the extreme point.

```
Plot3D[{ratio, 2}, {α, 0, 100}, {p, 2, 5}, AxesLabel → Automatic]
Plot3D[{ratio, 2}, {α, 0, 100}, {p, 2, 3}, AxesLabel → Automatic]
```




We observe the competitive ratio increases when $\alpha$ decreases for p between 2 and 3. However, for larger p the behavior changes and the competitive ratio even exceeds 2 for p larger 4. We apply the algorithm BEAT only for p less than 3, where the competitive ratio increases when $\alpha$ decreases. Thus, the adversary chooses the smallest feasible value for $\alpha$. For this value, we either have $\delta = 0$, if this is attained for $\alpha > 0$. Otherwise, the maximal ratio is attained for the limit of $\alpha$ going to 0. However, we observed in the plot above that $\delta = 0$ is always attained for $\alpha > 0$. Thus, the second case never occurs. The first case we have already considered above ($\delta = 0$). Thus we do not get a new bound on the ratio for $\delta$ being the extreme point.

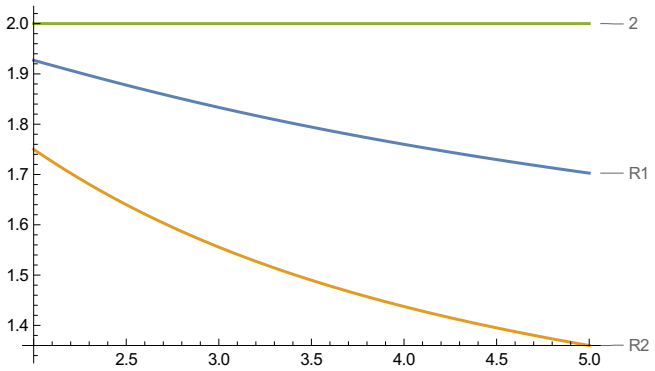
## Summary

We found two bounds on the competitive ratio: R1 and R2. We plot them to show R1 is always larger in

Appendix

the interval for p which we consider. Thus R1 is the bound we obtain.

```
Plot[{R1, R2, 2}, {p, 2, 5}, PlotLabels → Automatic]
```



# Analysis of the algorithm UTE

UTE is an algorithm for instances with uniform upper limit p for all jobs and the additional restriction that all precessing times are either 0 or p. It is parameterized by some function $\beta$. We show it has competitive ratio $\rho$ for such instances.

UTE behaves as follows: If the upper limit p is at most $\rho$, all jobs are executed without test. Otherwise, all jobs are tested. The first max{0, $\beta$} fraction of the jobs is executed immediately after its test. The other jobs are delayed, unless they have size 0.
The parameters $\beta$ and $\rho$ are

```
Clear[ρ, p, β, γ];
```

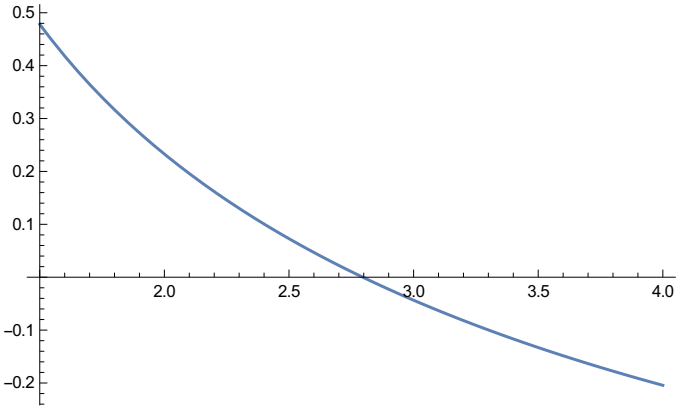
$$\beta\text{guess} = \frac{1 - p + p^2 - \rho + 2\,p\,\rho - p^2\,\rho}{1 - p + p^2 - \rho + p\,\rho};$$

$$\rho\text{guess} = \frac{1}{2}\left(1 + \sqrt{3 + 2\sqrt{5}}\right);$$

The adversary chooses p and a fraction $\gamma$, such that the last $\gamma$ fraction of the tests of UTE returns processing time 0 and all jobs tested before have processing time p.

By Proposition 1 we can assume p > $\rho$.

```
Plot[βguess /. {ρ → ρguess}, {p, 1.5, 4}]
```




We observe that $\beta$ is decreasing in p, until some root, which we call pstar. Also between 1.5 and pstar the value of $\beta$ does not exceed 1.

```
sol = Solve[βguess == 0, p]
N[sol /. {ρ → ρguess}]
```

$$\left\{\left\{p \to \frac{1 - 2\,\rho + \sqrt{-3 + 4\,\rho}}{2\,(1 - \rho)}\right\}, \left\{p \to \frac{-1 + 2\,\rho + \sqrt{-3 + 4\,\rho}}{2\,(-1 + \rho)}\right\}\right\}$$

$$\{\{p \to 0.357644\}, \{p \to 2.79608\}\}$$

The second root is the one that is relevant for us.

Appendix

```
pstar = p /. sol[[2]];
```

The sum of completion times of the optimal schedule can be described as :

```
OPT2 := n^2/2 (γ^2 + p * (1 - γ)^2 + 2 γ (1 - γ));
OPT1 := n/2 (γ + p (1 - γ));
OPT := OPT2 + OPT1;
```

We distinguish three cases depending on p and $\gamma$. By Proposition 1, we only have to consider p ≥ $\rho$.

## Case p ≤ pstar and g ≤ 1 - $\beta$

We treat the quadratic part of ALG and OPT separately from the linear part.

```
Clear[p, γ, ALG2];
ALG2 :=
  n^2/2 ((p + 1) β^2 + γ^2 + p (1 - β - γ)^2 + 2 (1 - γ + p β) γ + 2 (1 + p β) (1 - β - γ));
goal2 = 2/n^2 FullSimplify[ρ OPT2 - ALG2]
```

$$-2-\left(-2+\beta\right)\beta+\gamma^2-\left(-2+\gamma\right)\gamma\rho+p\left(-1+\gamma\left(-2\beta+\left(-2+\gamma\right)\left(-1+\rho\right)\right)+\rho\right)$$

The ratio is at most $\rho$ if goal is non-negative. Hence the adversary tries to minimize goal2 choosing p and $\gamma$, while the algorithm wants to maximize it or at least make it non-negative choosing $\beta$ and $\rho$. We show goal2 is convex in $\gamma$.

```
FullSimplify[D[D[goal2, γ], γ]]
```

$$2\left(-1+p\right)\left(-1+\rho\right)$$

Hence, the adversary chooses the extreme point.

```
sol = Solve[D[goal2, γ] == 0, γ]
N[sol /. {ρ → ρguess, p → ρguess, β → βguess /. {ρ → ρguess, p → ρguess}}]
```

$$\left\{\left\{\gamma\to\frac{-p+p\beta-\rho+p\rho}{\left(-1+p\right)\left(-1+\rho\right)}\right\}\right\}$$

$$\{\{\gamma\to 0.381966\}\}$$

The extreme point $\gamma$ is feasible, so we select it.

```
γ = γ /. sol[[1]];
FullSimplify[D[D[goal2, β], β]]
```

$$-2-\frac{2\,p^2}{\left(-1+p\right)\left(-1+\rho\right)}$$

Since goal2 is concave in $\beta$, the algorithm would like to choose an extreme point.

```
Clear[β];
sol = FullSimplify[Solve[D[goal2, β] == 0, β]]
N[sol /. {ρ → ρguess, p → ρguess}]
```

$$\left\{\left\{\beta \to \frac{1 - p + p^2 - (-1 + p)^2 \rho}{1 - \rho + p\,(-1 + p + \rho)}\right\}\right\}$$

$$\{\{\beta \to 0.286961\}\}$$

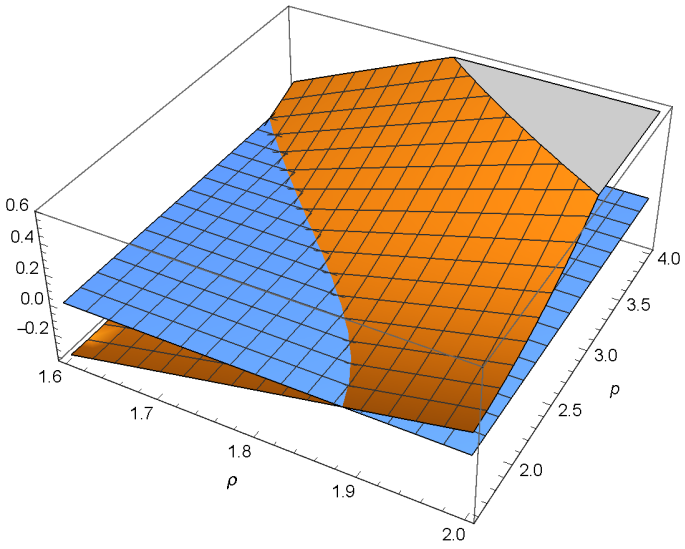
This is exactly the value we choose for $\beta$.

```
β = β /. sol[[1]];
```

goal2 now only depends on $\rho$ and p. We show that goal2 is increasing in p and in $\rho$. We need this function for goal2 later, so we call it goalsave.

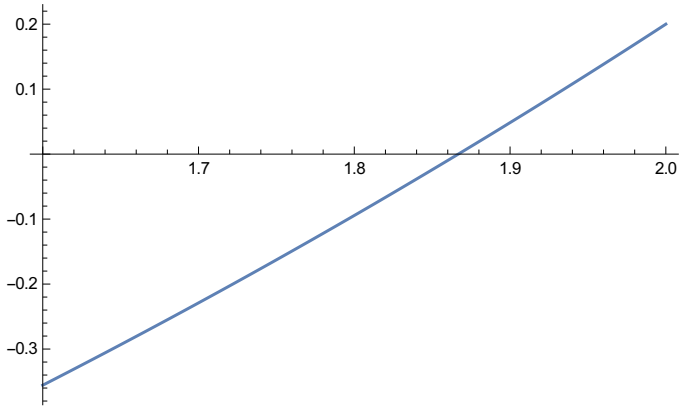
```
goalsave = FullSimplify[goal2];
```

```
Plot3D[{goal2, 0}, {ρ, 1.6, 2}, {p, 1.6, 4}, AxesLabel → Automatic]
```




Hence, the adversary chooses p = $\delta$. We plot goal2 with this p to show p = $\rho$ is the right choice.

```
p = ρ; Plot[goal2, {ρ, 1.6, 2}]
```



Appendix

To ensure goal2 is always positive, we need to choose the root of this goal as $\rho$.

```
sol = Solve[goal2 == 0, ρ]
N[%]
```

$$\left\{\left\{\rho \to \frac{1}{2}\left(1 - i\sqrt{-3+2\sqrt{5}}\right)\right\}, \left\{\rho \to \frac{1}{2}\left(1 + i\sqrt{-3+2\sqrt{5}}\right)\right\}, \left\{\rho \to \frac{1}{2}\left(1 - \sqrt{3+2\sqrt{5}}\right)\right\}, \left\{\rho \to \frac{1}{2}\left(1 + \sqrt{3+2\sqrt{5}}\right)\right\}\right\}$$

$$\{\{\rho \to 0.5 - 0.606658\, i\}, \{\rho \to 0.5 + 0.606658\, i\}, \{\rho \to -0.86676\}, \{\rho \to 1.86676\}\}$$

The forth root is the one relevant to us and it is the value we proposed as $\rho$guess.

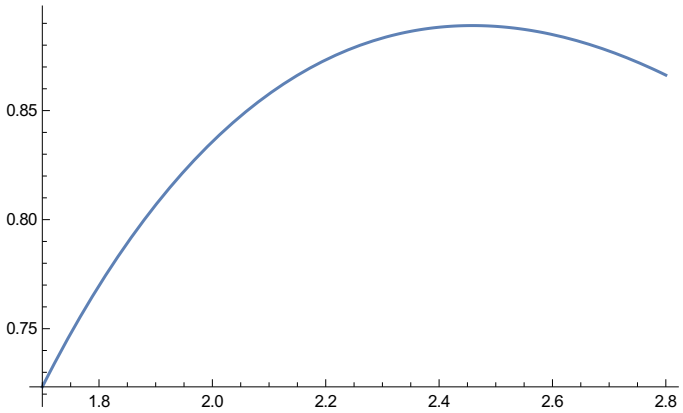
The linear algorithm cost is

```
Clear[p, γ]; ALG1 := n/2 ((p+1) β + γ + p (1-β-γ));
Apart[FullSimplify[goal1 = 2/n (ρ OPT1 - ALG1)], γ]
```

$$-(-1+p)\,\gamma\,(-1+\rho) + \frac{-1-p^3+\rho-p^2\,\rho+p^3\,\rho-p\,\rho^2+p^2\,\rho^2}{1-p+p^2-\rho+p\,\rho}$$

This is decreasing in $\gamma$, so we set $\gamma = 1 - \beta$ and show that for $\rho$guess goal1 is always positive.

```
γ = 1 - β; ρ = ρguess;
Plot[goal1, {p, 1.7, 2.8}]
```




## Case p>pstar

In this case max{$\beta$, 0} = 0 and UTE first tests and postpones the first 1 - $\gamma$ fraction of jobs (all of length p) and then tests and executes the remaining $\gamma$ fraction (all of length 0). Thus the algorithm cost ALG is

```
Clear[ρ, p, γ];
ALG := n^2/2 (γ^2 + p (1-γ)^2 + 2 (1-γ) γ) + n/2 (2 (1-γ) + γ + (p+1) (1-γ));
Apart[goal = FullSimplify[2/n (ρ OPT - ALG)], p]
```

$$-3+2\gamma-2\,n\,\gamma+n\,\gamma^2+p\,(-1+\gamma)\,(-1-n+n\,\gamma)\,(-1+\rho)+\gamma\,\rho+2\,n\,\gamma\,\rho-n\,\gamma^2\,\rho$$

The ratio is at most $\rho$ if goal ≥ 0.

```
FullSimplify[D[goal, p]]
```

$$(-1+n(-1+\gamma))(-1+\gamma)(-1+\rho)$$

Goal is increasing in p. Hence the worst case instance is realized at p=pstar.

```
p = pstar;
```

```
FullSimplify[D[D[goal, γ], γ]]
```

$$n\left(1+\sqrt{-3+4\rho}\right)$$

The goal is convex in $\gamma$. Hence the adversary will choose the extreme point of goal in $\gamma$.

```
sol = Solve[D[goal, γ] == 0, γ]
```

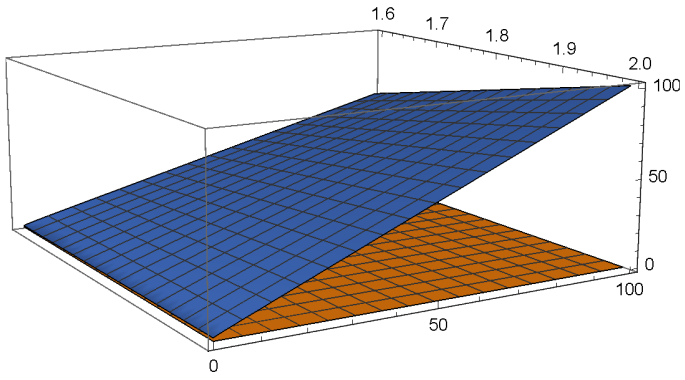
$$\left\{\left\{\gamma \to \frac{-5+2n+\sqrt{-3+4\rho}+2n\sqrt{-3+4\rho}}{2n\left(1+\sqrt{-3+4\rho}\right)}\right\}\right\}$$

```
γ = γ /. sol[[1]]; FullSimplify[goal]
```

$$\left(-11-2\rho+5\sqrt{-3+4\rho}+4n(1+n)(-1+\rho)\left(1+\sqrt{-3+4\rho}\right)\right)\Big/\left(4n\left(1+\sqrt{-3+4\rho}\right)\right)$$

We show goal is increasing in $\rho$ and n.

```
Plot3D[{0, goal}, {ρ, 1.6, 2}, {n, 1, 100}]
```




Furthermore, goal is positive for all feasible n and $\rho$, thus the ratio is at most $\rho$guess also in this case.

## Case p ≤ pstar and $\gamma > 1 - \beta$

In this case the algorithm tests and executes the first 1-$\gamma$ fraction of the jobs (of length p). We consider the linear and the quadratic terms in n separately. For the algorithm cost we have

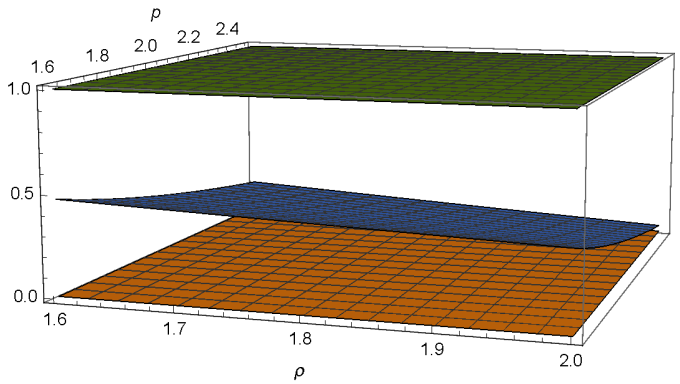
## Appendix

```
Clear[p, γ, ρ];
ALG2 := n^2 / 2 ((p + 1) (1 - γ)^2 + γ^2 + 2 (p + 1) (1 - γ) γ);
FullSimplify[goal2 = 2 / n^2 FullSimplify[ρ OPT2 - ALG2]]
```

$$-1-\left(-2+\gamma\right)\gamma\rho+p\left(-1+\gamma\right)\left(1+\gamma+\left(-1+\gamma\right)\rho\right)$$

We show the function $\beta$ is decreasing in p and $\rho$.

```
Plot3D[{0, β, 1}, {ρ, 1.6, 2}, {p, 1.6, 2.5}, AxesLabel → Automatic]
```




Thus $\beta$ is always smaller than the value it attains for p = $\rho$ and $\rho$ = $\rho$guess.

```
βmax = βguess /. {ρ → ρguess} /. {p → ρguess}
```

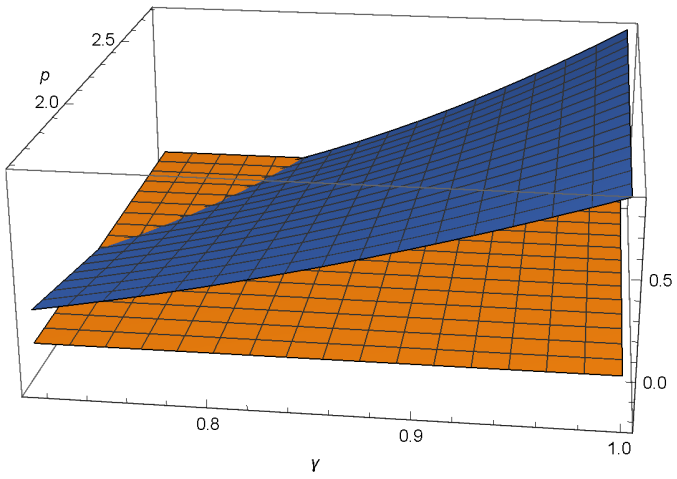
$$\left(-\sqrt{3+2\sqrt{5}}+\frac{3}{4}\left(1+\sqrt{3+2\sqrt{5}}\right)^2-\frac{1}{8}\left(1+\sqrt{3+2\sqrt{5}}\right)^3\right)\Big/\left(-\sqrt{3+2\sqrt{5}}+\frac{1}{2}\left(1+\sqrt{3+2\sqrt{5}}\right)^2\right)$$

```
N[%]
```

```
0.286961
```

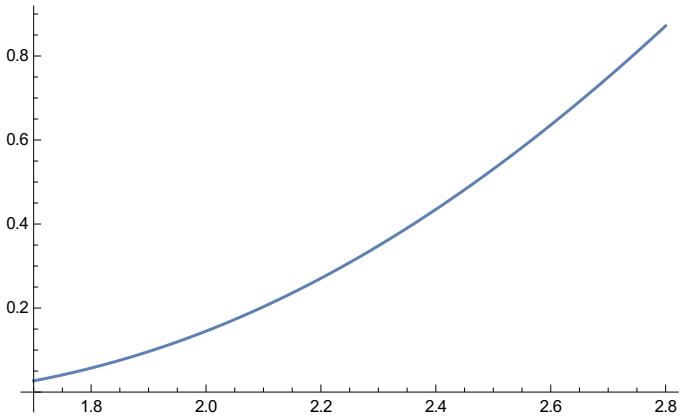
We set $\rho$ to the value we promise and show goal2 is decreasing in $\gamma$.

```
ρ = ρguess;
Plot3D[{0, goal2}, {γ, 1 - N[βmax], 1}, {p, 1.6, 2.8}, AxesLabel → Automatic]
```




Thus, we set $\gamma = 1 - \beta$ to show the goal is always positive.

```
γ = 1 - N[β]; Plot[goal2, {p, 1.7, 2.8}]
```




For the linear terms, we define goal1.

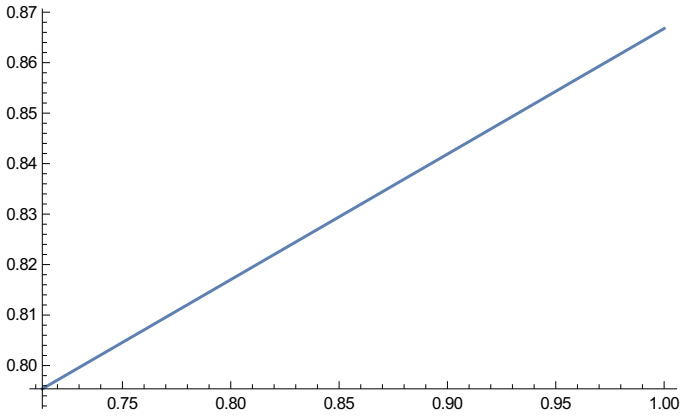
```
Clear[ρ, γ, p]; ALG1 = n/2 ((p + 1) (1 - γ) + γ);
Apart[goal1 = FullSimplify[2/n (ρ OPT1 - ALG1)], γ]
```

$$-1 - p + p\,\rho - \gamma\,(-p - \rho + p\,\rho)$$

The function goal1 is increasing in p, so we plot it for p = $\rho$.

Appendix

```
ρ = ρguess; p = ρ;
Plot[goal1, {γ, 1 - N[βmax], 1}]
```




This is always positive, as we desired.

# Performance for p of the lower bound instance

We take the upper limit p, that yields the general lower bound on deterministic instances and compute the competitive ratio UTE has for this instance.

```
Clear[n, p, γ, ρ];
ψ = N[Root[97 - 503 #1 + 1029 #1^2 - 1237 #1^3 + 566 #1^4 +
      984 #1^5 - 2521 #1^6 + 2948 #1^7 - 2130 #1^8 + 965 #1^9 - 250 #1^10 + 28 #1^11 &, 4]]
```

```
1.98962
```

In this case p is larger than $\rho$ and smaller than pstar. goalsave is goal for this case.

```
FullSimplify[goalsave]
```

$$\frac{-1-p^3+\rho+(-1+p)^2\,p\,\rho+(-1+p)\,\rho^2}{1-\rho+p\,(-1+p+\rho)}$$

goal always has to be positive, so we choose $\rho$ such that this is zero.

```
sol = Solve[0 == goalsave, ρ]
N[sol /. {p → ψ}]
```

$$\left\{\left\{\rho\to\frac{-1-p+2p^2-p^3+\sqrt{-3+6p-3p^2-6p^3+10p^4-4p^5+p^6}}{2\,(-1+p)}\right\},\right.$$
$$\left.\left\{\rho\to\frac{1+p-2p^2+p^3+\sqrt{-3+6p-3p^2-6p^3+10p^4-4p^5+p^6}}{2\,(1-p)}\right\}\right\}$$

```
{{ρ → 1.85519}, {ρ → -4.83465}}
```

The first solution is the only valid one and it almost matches the lower bound of 1.8546.