Diplom Thesis

Bad Guys are Rare: Probabilistic Analysis of an Elementary Dial-a-Ride Problem

Benjamin Hiller*

Ilmenau, 2004/07/05

supervised by:

Martin Dietzfelbinger
Sven O. Krumke

Ilmenau Technical University
Computer Science and
Automation Faculty
Gustav-Kirchhoff-Straße 1
D-98693 Ilmenau

Zuse Institute Berlin
Department Optimization
Takustraße 7
D-14195 Berlin-Dahlem

*inventory number: 2004-07-05/059/IN99/2239

Creating this thesis would have been much more inconvenient without great tools as LaTeX and its plethora of packages, KOMA-Script, METAPOST, GNU Emacs, GNU `make`, CVS, `gnuplot`, and last but not least Perl.

Acknowledgements

Although my name is the only one appearing on the title page, a lot of people have been directly or indirectly involved in the creation of this thesis.

First of all I want to thank my supervisors, Prof. Dr. Martin Dietzfelbinger and Prof. Dr. Sven O. Krumke. Thanks go to Martin Dietzfelbinger for supporting my work at ZIB by giving me the opportunity to work at the Studienarbeit and this thesis in Berlin. He and Prof. Dr. Manfred Kunde aroused my interest for this strange theoretical stuff. Sven O. Krumke always directed and motivated me, providing help and discussions when I needed them.

Im also very grateful to Jörg Rambau, Andreas Tuchscherer, and Tjark Vredeveld for reading a preliminary version of this thesis and giving many hints and suggestions for improvements. I really had a hard time incorporating them!

Further thanks go to Amin Coja-Oghlan who kindly answered any questions I had concerning the result that makes up the main part of this thesis. I am also indebted to Prof. Dr. Martin Grötschel for providing me with an opportunity to work as a trainee at ZIB, which was the starting-point for my stay there. I have to mention my other colleagues from the Optimization Department who have a part in the nice and inspiring atmosphere there.

Last but not least I want to thank my parents and my girlfriend Petra Gottstein for supporting my studies.

Contents

List of Algorithms

1 Introduction

In this thesis we study a certain simple problem from a general class of transportation problems known as *Dial-a-Ride problems*. Dial-a-Ride problems abstract transportation problems frequently arising in practice, both in industrial and service applications. The basic setting is the following: We control a set of *servers*, which travel along a *transportation network*. Furthermore, there are *requests for transportation*, i. e., our servers have to carry some goods or persons from a source location to a destination. Now we have to decide which requests to assign to each server and the exact order of service such that a certain optimization criterion is met. In general there are some constraints to these decisions, for example limited capacity of the servers.

Real-world applications covered by this general framework are for example:

Berlin's Telebus Telebus is Berlin's transportation service for handicapped people. Handicapped persons may request to be transported at arbitrary times between arbitrary locations. These requests are collected a day in advance and shall be scheduled to a fleet of vehicles (mini-busses), which is rent on a day-by-day basis. Clearly, the objective is to incur small cost for renting vehicles while ensuring a punctual service. In this case the transportation network is the road network of Berlin. See Borndörfer et al. [8] for details of how to tackle this complex large-scale problem.

Commissioning in high rack warehouses High rack warehouses play an essential role in modern logistics. They are used to store and retrieve larger quantities of many different goods. Typically, each high rack is operated by an elevator-like system, where the elevator travels along a rectangular grid, which constitutes the transportation network. Requests consist of a set of goods which have to be commissioned into one packaging unit for delivery to customers. A description of a concrete high rack warehouse system can be found in Hauptmeier's Diplom thesis [23].

The focus of this thesis is more limited. Many of the Dial-a-Ride problems arising in practice feature complex constraints, often interacting in a nontrivial way with

application-specific issues. The analysis of such intricate systems is very involved if not intractable. One therefore tries to consider simplified problems exhibiting the essential features. This theoretical approach is taken in this thesis.

1.1 Graph-theoretic Interpretation of Dial-a-Ride Problems

We now derive a formalization of Dial-a-Ride problems which will be the basis for the further discussion. This formalization is based on concepts from Graph Theory, see Section A.1 for a short introduction.

To simplify our graph-theoretic model and since we will later study a more specialized version anyway, we first make some further assumptions on the Dial-a-Ride problems under consideration. We assume that:

- There is only one server, which can serve at most one request at a time.
- All requests can be served by just traveling to a source location, picking up the object to transport and traveling to the destination location where the object is delivered; there are no other constraints.
- There is a distinguished location (called *depot*) from which the server unit starts and has to return to.
- Once the service of a request has started there must not be an interruption until the request is finished at the destination location (*non-preemptive transportation*).
- There are only finitely many interesting locations (i. e., source and destination locations) and we know the structure of the transportation network, especially the distances between the locations.
- The objective is the total completion time (also known as *makespan*), that is we want to finish all requests as early as possible. This is equivalent to minimizing the total travel distance.

These restrictions allow for the following abstract description: Each location is modeled by a vertex of a graph $G = (V, E)$ whose edges are possible interconnections between the locations. Furthermore we use an edge length function $c\colon E \to \mathbb{R}_{\geq 0}$ assigning each edge the length of the corresponding interconnection. A transportation

request is then simply an ordered pair (or arc) of locations. A feasible transportation is given by a directed closed walk (sequence of arcs) on the vertex set of our graph, which starts and ends at the depot vertex $o \in V$ and traverses each request arc exactly once. Note that in order to obtain such a feasible transportation we may have to add arcs connecting the destination location of a request with the source location of the next one. These arcs correspond to *empty rides* of the server. Notice that we do not allow splitting a request arc into successive arcs traversing one edge each, so the non-preemptive-transportations-requirement will be fulfilled. We call a feasible transportation a *tour* or *solution*. Our goal is to find a tour minimizing the total travel distance. Formally, we have the following optimization problem:

Definition 1.1 (Dial-a-Ride Problem DARP)

DARP

Instance: A (connected) graph $G = (V, E)$, called *underlying network*, a multiset A of arcs from $V \times V$, a distance function $d\colon E \to \mathbb{R}_{\geq 0}$ for the edges of G and a distinguished vertex $o \in V$.

Output: A closed walk starting and ending at o which traverses each $a \in A$ and has minimal length w. r. t. the *lifted distance function* $D\colon V \times V \to \mathbb{R}_{\geq 0}$, which assigns to each possible arc $(u, v) \in V \times V$ the length of a shortest path from u to v in G. More precisely, we are required to determine a tour $\mathcal{T}$ minimizing

$$\sum\nolimits_{(u,v)\in\mathcal{T}} D(u, v).$$

We say that an edge $\{u, v\}$ is *traversed by an arc* (u', v') if the path connecting u' and v' contains $\{u, v\}$.

In this thesis we will only deal with the case that the underlying transportation network is a tree. In the following two overview sections we remark on the known results for general graphs, too.

1.2 Offline Dial-a-Ride Problems: State of the Art and New Results

The DARP in its general form includes the Traveling-Salesman-Problem as an important special case and is thus NP-hard. Frederickson and Guan [18] show that the DARP is NP-hard, even for rather restricted classes of underlying networks such

as trees. An interesting fact is that the DARP can be solved in polynomial time if the graph G is a path, but it is NP-hard on a special tree called caterpillar which is essentially the path with n vertices [23, 4]. A systematic survey studying the complexity of many variants of Dial-a-Ride problems has been undertaken by de Paepe et al. [16, 15]. They were able to identify "maximal easy" and "minimal hard" problems as well as a small set of problems with unknown complexity status.

The standard way to deal with NP-hard problems is to look for *approximation algorithms*, which try to find a-close-to-optimal solution in polynomial time. For the case of DARP for general graphs a $\frac{9}{5}$-approximation algorithm was presented by Frederickson et al. [19]. An improved algorithm for trees proposed by Frederickson and Guan [18] has a performance ratio of $\frac{5}{4}$.

However, it has been observed that another algorithm proposed by Frederickson and Guan [18] called USE-MST performs even better in practice. In fact, most of the solutions generated by USE-MST were indeed optimal, whereas the performance guarantee of USE-MST is only $\frac{4}{3}$. This suggests to do *probabilistic analysis*: Instead of considering the behaviour of an algorithm w. r. t. worst-case instances, we are interested in its "typical" behaviour on instances drawn from a certain probability distribution.

This approach was taken by Coja-Oghlan et al.: They first showed that USE-MST solves asymptotically all instances optimal if it is run on a caterpillar [13] and later extended this result to general trees [12]. We were able to improve an intermediate result of the last-mentioned analysis which is presented in detail in Chapter 4.

Another common probabilistic technique in algorithm analysis is *average case analysis*: As in probabilistic analysis, one assumes that the considered DARP instances occur according to a probability distribution and shows that the behaviour of a certain algorithm is good on average. "Being good" may mean (expected) polynomial running time and / or better solution quality than is guaranteed for the worst case. The result shown by Coja-Oghlan et al. is stronger: The algorithm USE-MST has always polynomial running time (in fact, nearly-linear running time) and is optimal on almost all large instances, whereas an algorithm with good performance on average may be bad on a large part of the instances.

1.3 Online Dial-a-Ride Problems: State of the Art and New Results

We are also interested in *online versions* of the DARP modeling more appropriately the lack of information about future requests, which is an important feature of real-world applications. In environments encountered in practice there often is the following situation: While a server unit serves its transportation requests, new requests waiting for service become available. It is not possible to wait until the last request has arrived and then serve all the requests in an optimal way for the following reasons:

- The stream of requests may be very long or even (practically) infinite.
- There is no sufficient capacity to store the requests.
- Each request has to be served as fast as possible since other processes depend on its service.

Therefore the server has to serve the requests somehow while others arrive.

For online considerations, we will assume that the server travels at unit speed. (Equivalently, the edge distance function gives the time needed to traverse that edge.) It is not quite clear what objective to choose for the online problem; in fact, there are at least the following natural choices:

- Minimize the distance traveled by the server.
- Minimize the completion time, i. e., the time at which the server returns to the depot and the last request has been served (also known as *Makespan*).
- Minimize the (average) time a request is unserved, i. e., the so-called *flow time*.

We will mainly focus on the total travel distance, mainly because this objective is very similar to the offline objective and we hope to be able to exploit our knowledge of the offline problem. Both total travel distance and completion time are relatively easy to analyze. A problem with the completion time is that it does not make much sense in an environment where requests arrive at a steady rate since there will always be work to do. Clearly, the most interesting objective for such a system is the flow time since one is usually interested in serving requests as fast as possible. The flow time can thus be seen as a sort of "quality of service" criterion whereas the total travel distance corresponds to the cost of service.

The standard performance measure for online algorithms is the *competitive ratio*. An online algorithm is compared to an optimal offline algorithm knowing the entire request sequence in advance. The competitive ratio is the ratio of the online algorithm's cost compared to those of the offline algorithm. This is very similar to the approximation ratio for approximation algorithms.

Online versions of the DARP so far have been investigated for general graphs. Its is known that there is no competitive algorithm w. r. t. total travel distance. The algorithm SMARTSTART introduced by Ascheuer et al. [3] is 2-competitive w. r. t. completion time, which is optimal. Hauptmeier et al. [23, 24] showed that under appropriate restrictions to the request sequence the IGNORE-strategy leads to bounded average and maximum flow time.

The contribution of this thesis is to extend competitive analysis to the probabilistic setting: Analogously to the offline situation, we are interested in the competitive ratio of "typical" instances. We show a first such result stating that the IGNORE-strategy employing USE-MST is asymptotically optimal w. r. t. total travel distance if requests arrive rapidly, assuming a tree transportation network. Thus it is unlikely that a server controlled by IGNORE travels substantially further than the optimal distance.

1.4 Dial-a-Ride Problems on Trees

As mentioned before we will investigate the DARP on trees in this thesis. The original motivation for this restriction was the performance analysis of a large distribution center, whose pallet transportation system features some vertical elevators [1]. These were modelled by a path which was later extended to a caterpillar graph to allow modeling different acceleration and deceleration times.

From the theoretical point of view the DARP on trees is interesting, too. First of all, it has a particularly simple combinatorial structure. Secondly, the DARP is not thoroughly understood even for such elementary special cases. Furthermore there are techniques which allow the transfer of results obtained for trees to arbitrary graphs with a penalty factor of $\mathcal{O}(\log n)$ [5, 6, 17], although in some cases better direct results for general graphs are known.

In the remaining thesis the underlying transportation network will be a tree denoted by $T = (V, E)$. By convention, we always have $|V| = n$, $|E| = n - 1$ and $|A| = m$.

Moreover we will assume that the considered DARP instance has a certain struc-

ture, namely that the tree T does not contain leaves which are neither source nor destination vertex of a request nor depot. This assumption can be enforced via an adequate preprocessing.

Proposition 1.2 *Let $I = (T, A, d, o)$ be an arbitrary* DARP *instance. Then the instance $I' = (T', A, d, o)$ where T' is obtained from T by repeatedly deleting leaves, which are neither source vertex, destination vertex nor depot, has an optimal tour which is equivalent to an optimal tour of I with the same total length. Moreover, all edges of T' will be traversed by any optimal tour for I'.*

Proof. Clearly, an optimal tour $\mathcal{T}$ for I will not use edges with no requests starting or ending at the other side, so it can be directly translated to T', since none of the vertices $\mathcal{T}$ visits is deleted. On the other hand T' is just a connected subgraph of T, so any optimal tour for I' is an optimal tour for I.

To see that all edges of T' are traversed by any optimal tour, suppose edge $\{u, v\}$ violates this requirement for the optimal tour $\mathcal{T}$. The edge $\{u, v\}$ is a cut in T, so all vertices visited by $\mathcal{T}$ have to be on one side of $\{u, v\}$, say on the side of u. But now there have to be leaves on the side of v which are neither depot nor source or destination vertex of any request, which is a contradiction. □

1.5 Overview on this Thesis

We start by introducing an approximation algorithm called USE-MST for the DARP on trees in Chapter 2. The algorithm USE-MST exploits the combinatorial structure exhibited at the beginning of this chapter. The cornerstone of USE-MST is the so-called *balancing operation*, which "glues" many of the requests to closed subtours. It is explained how this operation can be implemented efficiently in linear time and how the resulting subtours are connected to form an approximate solution to the input DARP instance. The material of Chapter 2 is essentially taken from an article of Frederickson and Guan [18]. Finally we observe that USE-MST yields optimal solutions whenever the subtours generated by balancing constitute a star metric.

Before turning to probabilistic analyses we review the necessary Probability Theory in Chapter A. Special emphasis is on the techniques underlying the probabilistic analyses in the later chapters.

We already mentioned that even though DARP on trees is NP-hard it has been observed that many instances are solved optimally by USE-MST. A probabilistic result giving an asymptotic verification of this observation is the subject of Chapter 4.

This result has been obtained by Coja-Oghlan et al. [13, 12]. The basic idea is to show that most of the instances constitute a star metric after the balancing operation.

A key feature of the probability model used in the analysis is that the underlying network and the corresponding distance function are assumed to be fixed and only the requests are generated according to a probability distribution. The reason for this separation is that in practice the topology and the transportation cost of the transportation networks are fixed. In contrast, there are lots of requests which are in principle unpredictable but may exhibit some statistical structure.

We were able to improve an important intermediate result (Lemma 4.9): The proof of this lemma is now nearly straightforward and we get a better constant, too. Furthermore, the presentation is more accessible than in the original paper.

Chapter 5 is devoted to first steps of an probabilistic analysis of the online version of DARP on trees. First we extend the list random model by *release times*, indicating the time a request becomes known, to arrive at online models. We distinguish between release times chosen in a deterministic way (similar to worst-case-analysis) and release times generated by random *interarrival times*, that is, the time between two successive requests is a random variable with a given distribution. We choose the exponential distribution here, because it is relatively easy to analyze and often used for modeling queuing systems, which bear some resemblance to our online models.

The standard way of (theoretically) evaluating online algorithms, namely *competitive analysis*, is reviewed. Furthermore, we explain probabilistic versions of competitive analysis: *Randomized competitive analysis*, which is the standard for comparing randomized online algorithms, and a new notion of *probabilistic competitive analysis*.

In the final section we show a first probabilistic competitiveness result for the DARP on trees. The main idea is that if there are enough requests, relatively few balancing arcs will be needed since the requests match up quite well. This is complemented by the observation that the additional distance traveled by an online server is in the largest part due to balancing arcs.

The thesis ends with concluding remarks and directions for future research.

2 A Fast Approximation Algorithm for the Dial-a-Ride Problem on Trees

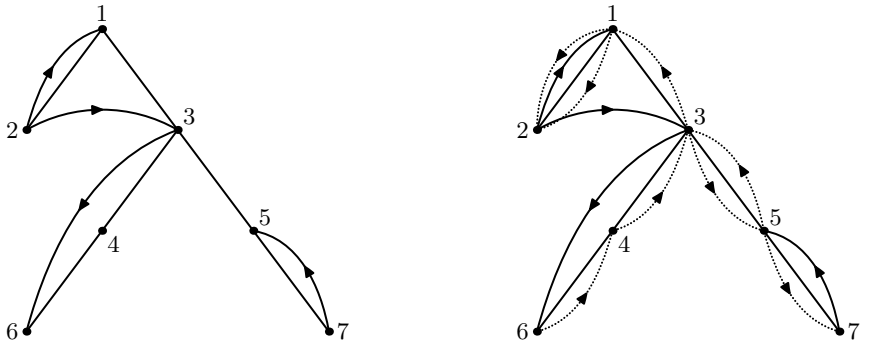
Recall that DARP is in P for paths but already NP-hard on trees. Thus approximation algorithms play a central role in DARP research. In this chapter we sketch an approximation algorithm for the DARP on trees which was proposed by Frederickson and Guan [18]. The algorithm USE-MST is not the best known one in terms of approximation quality but is introduced here because it generates optimal solutions quite often.

The emphasis is on the *balancing technique* which tries to add as many *unavoidable* empty rides as possible. It can also be used as a basis for similar approximation algorithms which has actually been done by Frederickson and Guan. They also show how to implement those algorithms efficiently and derive bounds on the approximation ratio. Their paper in part motivated Hauptmeier's Diplom thesis [23] which was a source for the material of this chapter, too.

The approximation ratio of USE-MST is $\frac{4}{3}$ and it can be implemented to run in time $\mathcal{O}\left(m + \mathcal{T}_{\mathrm{MST}}(q, n)\right)$ where q is the number of subtours after balancing and $\mathcal{T}_{\mathrm{MST}}(q, n)$ denotes the time needed to compute a MST of a graph with q vertices and n edges. To obtain this small running time we have to implement the balancing process carefully.

2.1 Basic Idea: Balancing Arcs

The main concept for the following approximation algorithm is that of a *balancing set*. Since a solution to the DARP is a closed walk and we are working on a tree, each edge $\{u, v\}$ has to be traversed from u to v as often as from v to u. Thus, we can safely (without increasing the objective value) augment our request set A with further *pseudo-requests* B that will ensure that this condition is satisfied, because



(a) Original instance.

(b) Instance with a balancing set.

Figure 2.1: A DARP instance. Throughout this thesis requests are indicated by arcs with an arrowhead in the middle. The dotted arcs are balancing arcs.

each solution has to traverse the arcs in B. One can think of the arcs in B as unavoidable empty rides.

As an example, consider the DARP instance in Figure 2.1(a). The edge $\{3, 1\}$ is traversed once from 1 to 3, namely by the request $(2, 3)$. It is traversed by no request times the other way round, so we know that the empty ride $(3, 1)$ is necessary to obtain a closed walk. Thus $(3, 1)$ belongs to the set B. Likewise, edge $\{1, 2\}$ is traversed twice from 2 to 1, so B gets two copies of $(1, 2)$. Figure 2.1(b) shows a set B for this instance.

To make this intuitive notion more precise we need some definitions. First of all, we consider a fixed DARP instance $I = (T, A, d, o)$ which is implicitly used in the remaining chapter. Notice that each $\{u, v\} \in E$ is a cut in the tree T and thus partitions the vertex set V in two sets $V(u)$ and $V(v)$ $(u \in V(u))$. For a tree T and a request multiset A, let $\Phi_{(T,A)}(u, v)$ denote the number of requests traversing $\{u, v\}$ from u to v; likewise, $\Phi_{(T,A)}(v, u)$ is the number of requests starting in $V(v)$ and ending in $V(u)$. If no tree and request set is specified, our generic tree T and request set A are implied.

Definition 2.1 (Balancing Set) An arc multiset B over $V \times V$ is called a *balancing set* for a DARP instance $(T = (V, E), A, d, o)$ if

$$\Phi_{(T,A\cup B)}(u, v) = \Phi_{(T,A\cup B)}(v, u) \qquad \forall \{u, v\} \in E. \tag{2.1}$$

We can compute a canonical balancing set $\overline{B}$ easily: Compute the number of times

an edge $\{u, v\}$ has to be traversed from u to v as

$$b(u,v) := \begin{cases} 1 & \Phi(u,v) = \Phi(v,u) = 0 \\ \Phi(v,u) - \Phi(u,v) & \Phi(v,u) > \Phi(u,v) \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

and construct $\overline{B}$ to contain $b(u, v)$ copies of (u, v) for all adjacent vertices u and v. The balancing set shown in Figure 2.1(b) is the set $\overline{B}$ for this instance.

Lemma 2.2 *$\overline{B}$ is a balancing set for the* DARP *instance $I = (T, A, d, o)$. Moreover, $\overline{B}$ is a subset of the arcs of* every *optimal tour.*

Proof. The fact that $\overline{B}$ is indeed a balancing set for the DARP instance $I = (T, A, d, o)$ follows directly from the definition of $b(u, v)$.

We still need to verify that the arcs in $\overline{B}$ are contained in every optimal tour for I. To this end, we check the cases of Equation (2.2) which defines $b(u, v)$ and thus $\overline{B}$.

The first case deals with edges which do not have to be traversed for satisfying the requests, but are used to connect subtours for some requests to bigger ones. Recall that our tree T does not contain leaves which are neither source vertex, destination vertex nor depot so all edges will be used by any tour (Proposition 1.2). For such edges, one arc for each direction has to be inserted.

In the second case there are more requests for transportation from $V(v)$ to $V(u)$ than from $V(u)$ to $V(v)$; the difference is just the number of times the server has to traverse edge $\{u, v\}$ from u to v without carrying an object since a solution is a closed walk.

The only remaining case is that there are some requests from $V(u)$ to $V(v)$ and fewer requests the other way round, in which $b(u, v)$ is zero. In fact there is no need to travel in that direction more often than there are requests in that direction. □

If we have a balancing set B for our input instance, we may be lucky and the digraph $(V, A \cup B)$ is strongly connected. In that case every Eulerian tour in that graph is an optimal tour, since it uses only arcs present in *all* possible tours. In general we have the following result.

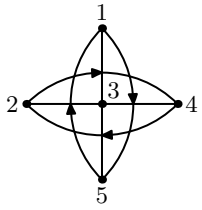
Lemma 2.3 *Every weakly connected component of $(V, A \cup B)$ is strongly connected and Eulerian.*

Proof. Let C be a weakly connected component of $(V, A \cup B)$. It suffices to show that C is Eulerian since this implies strong connectedness.

If we can show that $\delta^+(u) = \delta^-(u)$ for every vertex u in the vertex set of C it follows that C is Eulerian (see Lemma A.3). To see that $\delta^+(u) = \delta^-(u)$, we first replace every arc $(u, v) \in A \cup B$ by the corresponding directed path from u to v. Note that this affects $\delta^+(v)$ and $\delta^-(v)$ in the same way, i. e., they increase by the same value. Now suppose there is a u with $\delta^+(u) > \delta^-(u)$ (or vice versa). This implies that there is a vertex v such that there is an arc (u, v) but no corresponding arc (v, u). This means that condition (2.1) is not satisfiead for edge $\{u, v\}$, contradicting the definition of a balancing set. □

Corollary 2.4 *If the graph $(V, A \cup B)$ is strongly connected every Eulerian tour of $(V, A \cup B)$ is an optimal solution for the* DARP *instance $I = (T, A, d, o)$.* □

As mentioned above, every edge in T will be traversed so one might think that the digraph $(V, A \cup B)$ is always connected. The problem is that a request arc must not be broken in a sequence of edge-by-edge-arcs since transportation shall be non-preemptive. The following example instance shows that after balancing there may be more than one strongly connected component (here: $B = \emptyset$):



As we have seen, the graph $(V, A \cup B)$ may decompose into several strongly connected components. If such a component contains the depot vertex or at least one vertex incident with a request we call it *nontrivial*, *trivial* otherwise. In order to obtain a tour through all requests it remains to connect the subtours contained in the nontrivial components. The trivial components need not be visited, of course.

We want to employ minimum spanning tree algorithms to get short connecting arcs. It is well-known that this can be done in nearly linear time. However, our balancing set $\overline{B}$ may be as large as $\Omega(mn)$ (see [18, 4]) and thus needs at least $\Omega(mn)$ time to be generated. Since we are interested in a fast overall algorithm we strive to compute a balancing set in linear time.

In the next section we will explain how to compute a balancing set B in time $\mathcal{O}(n + m)$. The method for connecting nontrivial components via minimum spanning trees is given in Section 2.3. This is the last piece of the algorithm USE-MST which will be discussed in this section, too. Finally, the last section explains an important special case where USE-MST is indeed optimal.

Algorithm 2.1 Algorithm BALANCE for determining a balancing set.

BALANCE (T, A)

Input: Tree $T = (V, E)$, request multiset A.

Output: A balancing set for T and A.

1 Choose an arbitrary vertex $r \in V$ and root the tree T with respect to r, that is compute the parent vertex for each vertex.
2 Compute $b(u, v)$ and $b(v, u)$ for each $\{u, v\} \in E$ using algorithm COMPUTE-B.
3 Set $B := \emptyset$. For each $\{u, v\} \in E$ with $b(u, v) > 0$ $[b(v, u) > 0]$ add an arc (u, v) $[(v, u)]$ to B and decrease $b(u, v)$ $[b(v, u)]$ by one.
4 Add a new vertex r' and an edge $\{r', r\}$ to T. Set $b(r', r) = b(r, r') = 0$.
5 $B := B \cup$ BALANCING-ARCS-UP(r', r)
$B := B \cup$ BALANCING-ARCS-DOWN(r', r)
return B

2.2 Computing a Balancing Set of Arcs in Linear Time

When looking for a faster algorithm to compute a balancing set B, we have to keep in mind the following requirements and goals:

1. All arcs in B have to be traversed in any optimal tour (i. e., we do not give away anything by balancing).
2. B can be computed efficiently (should contain few arcs).
3. $(V, A \cup B)$ has not more nontrivial components than $(V, A \cup \overline{B})$.

In order to keep the number of arcs small, we use the following idea: If $\overline{B}$ contains a sequence of arcs $(v_1, v_2), (v_2, v_3), \ldots, (v_{l-1}, v_l)$, we may substitute those arcs by (v_1, v_l). Notice that any tour traversing the original arc sequence is equivalent to a tour traversing the arc (v_1, v_l) instead, so this construction meets requirement 1. The resulting multiset is again a balancing set with possibly more nontrivial components. To avoid this, our B will contain a submultiset of $\overline{B}$ such that no more nontrivial components arise.

Algorithm 2.1 shows the top-level-structure of BALANCE which computes a balancing set B from a tree T and a request multiset A. It uses two kinds of subroutines COMPUTE-B and BALANCING-ARCS-UP (BALANCING-ARCS-DOWN is similar to BALANCING-ARCS-UP). Step 1 roots the tree with respect to an arbitrarily chosen vertex r, which is an algorithmic trick to tackle the efficient computation of $b(u, v)$ in

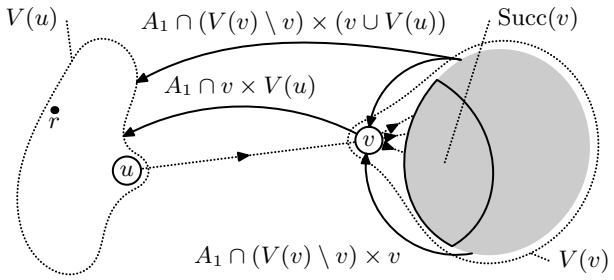


Figure 2.2: Situation when computing $\Phi(v, u)$ using A_1. Each edge $\{u, v\}$ is a cut in T, thus partioning the vertices in components $V(u)$ and $V(v)$. We can now count $\Phi(v, u)$ as follows: A request in A_1 traversing $\{u, v\}$ from v to u starts either in $V(v) \setminus \{v\}$ or in v. Therefore $\Phi(v, u)$ equals the sum of the number of requests starting in v and $V(v) \setminus \{v\}$ minus those starting in $V(v) \setminus \{v\}$ and ending in v.

step 2 as well as the balancing in step 5. Step 3 ensures that the constructed balancing set B has no more nontrivial components than $\overline{B}$ by adding one copy of each arc in $\overline{B}$. We still need to add the equivalent of the remaining arcs from $\overline{B}$ which is done in step 5. Those arcs are added by BALANCING-ARCS-UP and BALANCING-ARCS-DOWN, which do shortcut longer sequences of adjacent arcs in the way explained above. Again, the addition of an auxiliary root in step 4 is an algorithmic aid to avoid a special treatment of the root r in BALANCING-ARCS-UP and BALANCING-ARCS-DOWN.

We will later see that the overall running time of algorithm BALANCE is $\mathcal{O}(n + m)$, which is also the number of arcs in the generated balancing set B. In order to achieve this run time, the subroutines COMPUTE-B and BALANCING-ARCS-UP have to be designed carefully.

2.2.1 Computing the Balancing Defect $b(u, v)$

We want to compute the $b(u, v)$ via Equation (2.2), so we need to compute the $\Phi(u, v)$ first.

The basic idea for doing this is the following (see Figure 2.2): Suppose the input tree T has been rooted with respect to a root node r. Let A_1 be the multiset of all request-arcs directed towards the root and fix an arc (u, v) of our tree (this corresponds to an edge $\{u, v\}$ of the original tree and has been directed away from the root by the rooting process). Observe that $\Phi(v, u)$ depends only on arcs directed

towards the root. All arcs contributing to $\Phi(v,u)$ originate either in $V(v) \setminus \{v\}$ or in v. Arcs starting in $V(v) \setminus \{v\}$ contribute only to $\Phi(v,u)$, if they do not end in v. This leads to the formula

$$\begin{aligned}\Phi(v,u) = {} & |A_1 \cap (V(v) \setminus v) \times (v \cup V(u))| + |A_1 \cap v \times V(u)| \\ & - |A_1 \cap (V(v) \setminus v) \times v|.\end{aligned}$$

However, requests to be transported from $V(v) \setminus \{v\}$ to $V(u) \cup \{v\}$ have to pass through the set of v's successors $\mathrm{Succ}(v)$, since T is a tree. Using some basic graph theoretic notation we then have

$$\Phi(v,u) = \sum\nolimits_{w \in \mathrm{Succ}(v)} \Phi(w,v) + \delta^+_{(V,A_1)}(v) - \delta^-_{(V,A_1)}(v).$$

We have arrived at a recursive formula for $\Phi(v,u)$. The basis for this recursion is the case when v is a leaf. The formula then collapses to

$$\Phi(v,u) = \delta^+_{(V,A_1)}(v),$$

since $\delta^-_{(V,A_1)}(v)$ is zero. A similar recursion can be derived for the multiset A_2 of requests directed away from the root.

We assumed so far that each request is either directed towards the root or away from it. However, in general there will be requests that go some way towards the root and later away from it. The solution to this problem is to replace such request arcs by two arcs, one directed towards the root and one directed away from it (this replacement is done only for the computation of the $b(u,v)$).

We are now ready to give the algorithm COMPUTE-B (see Algorithm 2.2), which uses two subroutines COMPUTE-Φ-UP and COMPUTE-Φ-DOWN. These compute the values $\Phi(u,v)$ bottom-up in the way explained above, proceeding in depth-first manner.

Step 1 of Algorithm 2.2 does the replacement of request arcs which are not directed entirely towards or away from the root. The rest of the algorithm works as already sketched.

Lemma 2.5 *Algorithm 2.2 correctly computes $b(u,v)$ in time $\mathcal{O}(n+m)$.*

Proof. The correctness follows from the construction of the algorithm. It remains to discuss the running time.

The nearest common ancestor of two vertices can be found in time $\mathcal{O}(1)$ after a preprocessing taking time $\mathcal{O}(n)$ (see [22, 34]), so the substitution loop starting in

Algorithm 2.2 Algorithm COMPUTE-B for computing $b(u, v)$.

COMPUTE-B (T, A)

Input: A rooted tree T with root r, request multiset A.

Output: $b(u, v)$ and $b(v, u)$ for each pair of adjacent vertices u, v.

```
1  for all (u, v) ∈ A do
2    Let w be the nearest common ancestor of u and v.
3    if w ≠ u and w ≠ v then
4      Replace (u, v) by (u, w) and (w, v).
5    end if
6  end for
7  Partition A into multisets A_1 and A_2, containing requests directed towards r and
   away from r, respectively.
8  Precompute all δ⁺_(V,A_1)(v), δ⁻_(V,A_1)(v) and δ⁺_(V,A_2)(v), δ⁻_(V,A_2)(v) for all v ∈ V.
9  for all v ∈ Succ(r) do
10   Φ(v, r) := COMPUTE-Φ-UP(r, v)
11 end for
12 for all v ∈ Succ(r) do
13   Φ(r, v) := COMPUTE-Φ-DOWN(r, v)
14 end for
15 Compute all b(u, v) and b(v, u) via Equation (2.2).
   return b(u, v) and b(v, u)
```

COMPUTE-Φ-UP (u, v)

Input: An arc (u, v) of the tree T.

Output: $\Phi(v, u)$ and all $\Phi(v', u')$ for arcs (u', v') in the subtree below v.

```
1  Φ(v, u) := δ⁺_(V,A_1)(v) − δ⁻_(V,A_1)(v)
2  for all w ∈ Succ(v) do
3    Φ(w, v) := COMPUTE-Φ-UP(v, w)
4    Φ(v, u) := Φ(v, u) + Φ(w, v)
5  end for
   return Φ(v, u)
```

COMPUTE-Φ-DOWN (u, v)

Input: An arc (u, v) of the tree T.

Output: $\Phi(u, v)$ and all $\Phi(u', v')$ for arcs (u', v') in the subtree below v.

```
1  Φ(u, v) := δ⁻_(V,A_2)(v) − δ⁺_(V,A_2)(v)
2  for all w ∈ Succ(v) do
3    Φ(v, w) := COMPUTE-Φ-DOWN(v, w)
4    Φ(u, v) := Φ(u, v) + Φ(v, w)
5  end for
   return Φ(u, v)
```

line 1 takes time $\mathcal{O}(n+m)$. The partitioning step and the computation of the degrees needs time $\mathcal{O}(m)$. The subroutines COMPUTE-Φ-UP and COMPUTE-Φ-DOWN need only constant time at each vertex and visit each vertex once, resulting in time $\mathcal{O}(n)$. Putting all $\Phi(u,v)$ together to get the resulting $b(u,v)$ can be done with $\mathcal{O}(m)$ steps. □

2.2.2 Contracting Balancing Arcs

We need to describe how the subroutines BALANCING-ARCS-UP and BALANCING-ARCS-DOWN work. Recall that the task of these subroutines is to create balancing arcs according to the modified values of $b(u,v)$ and $b(v,u)$. The generated arcs have to be shortcut versions of longer successive arc sequences in $\overline{B}$.

We explain BALANCING-ARCS-UP in more detail. Each vertex v with $b(v,u) > 0$ (u is the parent of v) will be called *unsatisfied initial vertex* since arcs of the form (v,w) where w is an ancestor of v have to be generated in order to balance $\{u,v\}$. The algorithm computes for each vertex v a list $l(v)$ of unsatisfied initial vertices in the subtree with root v.

The general goal for BALANCING-ARCS-UP is to satisfy a vertex *as late as possible*, that is to generate *long* balancing arcs. Since we are considering balancing arcs directed towards the root this means that arcs starting deep down the tree should end near the root. Suppose we know the list l, which is the concatenation of all $l(w)$, where w is a successor of v. Then there are the following cases:

Case 1: v is a leaf, $l = \emptyset$
v has to be the initial vertex for $b(v,u)$ balancing arcs, so $l(v)$ contains exactly $b(v,u)$ copies of v.

Case 2: v is not a leaf; $b(v,u) > |l|$
We need more balancing arcs with initial vertex in the subtree of v than there are already in the list l, so we have to add $(b(v,u) - |l|)$ copies of v to l to obtain $l(v)$.

Case 3: v is not a leaf; $b(v,u) = |l|$
There are already exactly as many unsatisfied initial vertices in l as are needed to balance edge $\{u,v\}$, so $l(v) = l$.

Case 4: v is not a leaf; $b(v,u) < |l|$
We need fewer balancing arcs with initial vertex in the subtree of v than there are already in the list l, so $(|l| - b(v,u))$ arbitrary vertices of those have to be

Algorithm 2.3 Algorithm BALANCING-ARCS-UP for computing contracted balancing arcs directed towards the root.

BALANCING-ARCS-UP (u, v)

Input: Edge $\{u, v\} \in E$

Output: A multiset B of balancing arcs for edges $\{u, v\}$ with $b(v, u) > 0$ ending in v or its subtree. Furthermore, computes list $l(v)$.

```
1  B := ∅
2  if v is a leaf then
3     l(v) consists of b(v, u) copies of v.
4  else
5     l(v) is an empty list.
6     for all w ∈ Succ(v) do
7        B := B ∪ BALANCING-ARCS-UP(v, w)
8        Add l(w) to list l(v).
9     end for
10    if b(v, u) > |l(v)| then
11       Add (b(v, u) − |l(v)|) copies of v to l(v).
12    end if
13    if b(v, u) < |l(v)| then
14       Let k := |l(v)| − b(v, u) and v_1, ..., v_k be the first k vertices in l(v).
15       Add arcs (v_i, v) to B, 1 ≤ i ≤ k.
16       Delete v_1, ..., v_k from l(v).
17    end if
18 end if
   return B
```

satisfied by a balancing arc ending in v. The remaining unsatisfied vertices form the list $l(v)$.

Notice that only in the last case balancing arcs are generated.

The algorithm BALANCING-ARCS-UP (see Algorithm 2.3) will traverse the tree in a depth-first-search way similar to COMPUTE-Φ-UP to compute the list l before computing $l(v)$. We start with our artificial edge $\{r', r\}$ (see step 5 in Algorithm 2.1). Observe that since $b(r, r') = 0$ all vertices which are still unsatisfied after all successors of r have been traversed will be connected to r by balancing arcs.

Lemma 2.6 *The multiset of balancing arcs generated by Algorithm 2.3 has the property that it contains exactly $b(v, u)$ arcs that traverse an edge $\{u, v\}$ from v to u.*

Analogously, the similar algorithm BALANCING-ARCS-DOWN *adds exactly $b(u, v)$*

arcs traversing $\{u, v\}$ *from* u *to* v.

Proof. We know from the above discussion that $|l(v)| = b(v, u)$. We also noted that because of $b(r, r') = 0$ we have $l(r) = 0$, which tells us that every vertex which was once unsatisfied will be satisfied eventually. In the subtree rooted at v are exactly $|l(v)| = b(v, u)$ unsatisfied initial vertices, which lead to exactly $b(v, u)$ balancing arcs traversing $\{u, v\}$ from v to u. □

2.2.3 Running Time of the Balancing Algorithm

Theorem 2.7 *The algorithm* BALANCE *(Algorithm 2.1) computes a balancing set* B *with* $\mathcal{O}(n + m)$ *arcs in time* $\mathcal{O}(n + m)$.

Proof. The correctness follows from Lemmas 2.5 and 2.6.

In order to analyze the running time, let us first count the size of B. Step 3 adds at most $2n - 2$ balancing vertices, namely for each edge an arc in both directions.

To count the number of balancing arcs generated by BALANCING-ARCS-UP and BALANCING-ARCS-DOWN is more difficult. At the end of BALANCE the graph $(V, A \cup B)$ consists of several strongly connected Eulerian components D_i. Let k_i denote the number of arcs in D_i and let M be the set of arcs inserted by BALANCING-ARCS-UP.

Consider an Euler tour of D_i: There cannot be two consecutive arcs $(u, v), (v, w) \in M$ in that Euler tour since otherwise BALANCING-ARCS-UP would have generated an arc ending in v and another one starting in v, which is impossible (see steps 10 and 13 in Algorithm 2.3). Consequently, there cannot be more than $\lfloor \frac{k_i}{2} \rfloor$ arcs in D_i which have been added by BALANCING-ARCS-UP. A similar argument holds for BALANCING-ARCS-DOWN. In other words, BALANCING-ARCS-UP and BALANCING-ARCS-DOWN add at most as many arcs as have been inserted in step 3 of BALANCE (Algorithm 2.1) to ensure that the number of components does not increase. This shows $|B| \in \mathcal{O}(n + m)$.

The running time of BALANCE can be bounded as follows. Rooting the tree can be done in time $\mathcal{O}(n)$. Lemma 2.5 states that the computation of the $b(u, v)$ does not need more than $\mathcal{O}(n + m)$ steps. The preprocessing for the final balancing (steps 3 and 4) again takes time $\mathcal{O}(n)$. All we need to show yet is that BALANCING-ARCS-UP and BALANCING-ARCS-DOWN run in time $\mathcal{O}(n + m)$.

Notice that for each vertex v, BALANCING-ARCS-UP(u, v) is called exactly once (u is the direct predecessor of v). The time spent for traversing the tree is $\mathcal{O}(n)$, since at each vertex $\mathcal{O}(\delta_T(v))$ steps are needed. Since every vertex added to the list $l(v)$ is deleted to create a balancing arc and only $\mathcal{O}(n + m)$ balancing arcs are generated in

total, the total time to maintain the lists is also $\mathcal{O}(n+m)$, because each list operation can be done in constant time. This holds analogously for BALANCING-ARCS-DOWN.□

2.3 Connecting Nontrivial Components

As mentioned before, there may be more than one component after balancing (the original input may be balanced also). It thus remains to connect the resulting strongly connected Eulerian components at minimal distance to obtain an overall solution.

As it turns out, the complexity of the DARP on trees stems from this connection step. Frederickson and Guan [18] show the NP-hardness of the DARP on trees by reducing a decision variant of STEINERTREE to the decision version of DARP.

STEINERTREE

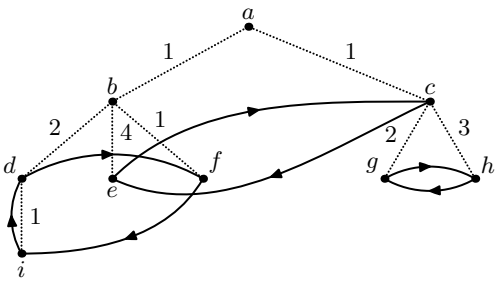
Instance: A graph $G = (V, E)$, an edge length function $d\colon E \to \mathbb{R}_{\geq 0}$ and a subset $U \subseteq V$. (The vertices U and $V \setminus U$ are called *terminals* and *Steiner points*, respectively.)

Output: A subtree S of G with minimum length $d(S)$ spanning at least the vertices in U. (A subtree S of G spanning at least U is called a *Steiner tree*.)

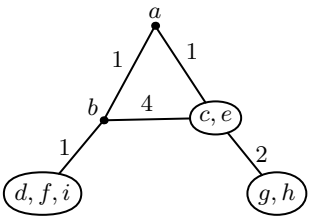
We will not give the NP-hardness proof here but motivate where the connection to Steiner tree problems comes from.

Consider a balanced DARP instance $I = (T, A \cup B, c, o)$ with a possibly empty balancing set B. In the sequel we will not distinguish anymore between original request arcs from A and pseudo-request arcs in B introduced by the balancing process and simply write $\hat{A} := A \cup B$. The so-called *arc-identified graph H for instance I* is the graph arising if in $(T, \hat{A})$ all vertices connected by arcs are identified, i. e., all strongly connected components are contracted to a single vertex. It is helpful to think of the vertices of H as being labeled by the set of original vertices in T. Figure 2.3(b) shows the arc-identified graph for the instance displayed in Figure 2.3(a).

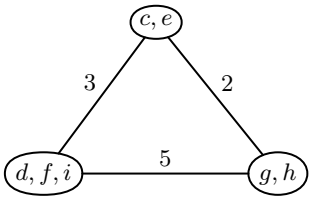
Connecting the nontrivial components of our balanced instance I can be done by solving a STEINERTREE instance: In the arc-identified graph H, we have two kinds of vertices: Those whose labels contain the start or destination vertex of some request or the depot o (and thus represent the nontrivial components to be connected) form the terminal set U; all other vertices are Steiner points. If we can find a Steiner tree S of H, we have to add arcs (u, v) and (v, u) for each edge $\{u, v\}$ used by S to our arc



(a) Instance with a balanced request set.



(b) The arc-identified graph for the instance.



(c) The component graph for the instance.

Figure 2.3: A DARP instance and its arc-identified and component graph. Notice that in the original instance every edge is used at least once in each direction.

set to get one strongly connected Eulerian component in the original instance, which corresponds to an optimal solution.

Theorem 2.8 *Let $I = (T, \hat{A}, d, o)$ be a balanced* DARP *instance, H its arc-identified graph as above and define the digraph $D := (V, \hat{A})$. An optimal Steiner tree S^* is related to an optimal tour $\mathcal{T}$ for I by*

$$\text{DARP}(I) = d(\mathcal{T}) = d(\hat{A}) + 2d(S^*).$$

In other words: An optimal Steiner tree S^* can be used to obtain an optimal tour $\mathcal{T}$.

Proof. First note that each vertex of H corresponds to a component of D.

$d(\hat{A}) + 2d(S^*) \geq d(\mathcal{T})$

Let S be a subtree of H with length $d(S)$ such that S spans at least U. Consider the directed graph D' on vertex set V which consists of the arcs from $\hat{A}$ and arcs (u, v) and (v, u) for each edge $\{u, v\}$ contained in S. This graph is Eulerian because $\hat{A}$ is balanced and thus degree-balanced and the additional arcs also keep degrees balanced. Therefore, D' admits an Euler tour $\mathcal{T}$ which is a solution for I since all

requests $\hat{A}$ as well as the depot o belongs to nontrivial components represented by the vertices of H, which are connected by S. The length of an optimal tour $\mathcal{T}$ is thus bounded by

$$d(\mathcal{T}) \leq d(\hat{A}) + 2d(S)$$

for any subtree S.

$d(\mathcal{T}) \geq d(\hat{A}) + 2d(S^*)$

Let $\mathcal{T}$ be an optimal tour for I with length $d(\mathcal{T})$. We know that every tour has to traverse all arcs in $\hat{A}$ which are the arcs of D so we can write $d(\mathcal{T}) = d(\hat{A}) + 2x$. Denote by E' the set of edges of T traversed by $\mathcal{T}$ without servicing a request and let E'' be the set of edges of H corresponding to E'. The induced graph $H[E''] =: S_0$ spans the set U since E' connected all nontrivial components and thus has a subgraph S which is a tree and still spans U.

Since $\mathcal{T}$ is an optimal tour it does not traverse an edge $\{u, v\}$ of S more than once in each direction without servicing a request. This can be seen by the following construction: Let $\{u, v\}$ be an edge of T which is traversed by $\mathcal{T}$ at least twice from u to v without servicing a request. Assume that $\mathcal{T}$ traverses $\{u, v\}$ in that direction r times. Split all requests (u_i, v_i), $1 \leq i \leq r$, using $\{u, v\}$ from u to v in arcs (u_i, u), (u, v) and (v, v_i). As $\hat{A}$ is balanced there are r requests using that edge in the other direction which can be split similarly. If we now remove $(r-1)$ copies of (u, v) and (v, u) the graph remains connected and Eulerian, because afterwards every indegree equals every outdegree as before. If the length of $\{u, v\}$ is positive this deletion decreases the length, contradicting the fact that $\mathcal{T}$ was an optimal tour. Otherwise we get a tour with the same length which traverses $\{u, v\}$ once in each direction.

Now that we are convinced that each edge of S is traversed at most twice we see that

$$d(S) \leq d(S_0) = \frac{d(\mathcal{T}) - d(\hat{A})}{2} = x,$$

which is equivalent to $d(\mathcal{T}) \geq d(\hat{A}) + 2d(S^*)$. □

We sketch an approximation algorithm for the connection of the components. It uses a minimum spanning tree of the *component graph* of our balanced instance I. The minimum spanning tree problem is

MINIMUM SPANNING TREE (MST)

Instance: A graph $G = (V, E)$ and an edge length function $d\colon E \to \mathbb{R}_{\geq 0}$.

Output: A subtree T of G with minimum length $d(T)$ spanning all vertices of V.

In a sense, the minimum spanning tree problem is the special case $U = V$ of the Steiner tree problem. This difference is essential: There are polynomial time algorithms for solving MST whereas SteinerTree is NP-hard which justifies using an approximation algorithm.

The *component graph H' for instance I* is a weighted complete graph, whose vertex set is the set of nontrivial components of I, i. e., the set U mentioned above. The weight $D_U \colon U \times U \to \mathbb{R}_{\geq 0}$ of an edge is the length of a shortest path connecting both nontrivial components (Note that paths from all vertices in the first to all vertices in the second component are considered.). The component graph for our above example instance is shown in Figure 2.3(c). Let S' be a minimum spanning tree of H'. To obtain a set of linking arcs between the nontrivial components we can take S' as an approximation to S, add an arc for each edge used by S' to the graph $(V, A \cup B)$ which is then Eulerian. Any Euler tour is a solution to our instance I. The steps are summarized in Algorithm 2.4. Frederickson and Guan [18] show how to implement this efficiently and prove the following result.

Theorem 2.9 *[18] For every* Darp *instance $I = (T, A, c, o)$ the algorithm* use-MST *computes a tour which is at most $\frac{4}{3}$ times as long as an optimal one. The running time is $\mathcal{O}\left(m + \mathcal{T}_{\mathrm{MST}}(q, n)\right)$ where q is the number of nontrivial components and $\mathcal{T}_{\mathrm{MST}}(q, n)$ denotes the time needed to compute a* MST *of a graph with q vertices and n edges.* □

Remark The exact complexity of the minimum spanning tree problem is not known. It is known, however, that $\mathcal{T}_{\mathrm{MST}}(n, m) \in \Omega(m)$ and $\mathcal{T}_{\mathrm{MST}}(n, m) \in \mathcal{O}\left(m\alpha(m, n)\right)$ (see [10, 31]), where $\alpha(m, n)$ is an inverse of the Ackermann function and n and m are the number of vertices and edges, respectively. Since $\alpha(m, n)$ grows extremely slowly it is legitimate to say that use-MST is a nearly-linear time algorithm. Curiously there is an optimal algorithm, i. e., one with running time $\mathcal{O}\left(\mathcal{T}_{\mathrm{MST}}(n, m)\right)$, but the function $\mathcal{T}_{\mathrm{MST}}(n, m)$ has not yet been determined [32].

2.4 A Special Case Where use-MST is Optimal

There is an interesting special case for the SteinerTree-Problem, in which the MST is not longer than the optimal Steiner tree. Notice that the graph $G = (V, E)$ with edge length function $d \colon E \to \mathbb{R}_{\geq 0}$ in the input of SteinerTree can be viewed

Algorithm 2.4 Algorithm USE-MST for computing an approximate solution to the DARP on trees.

USE-MST (T, A, c, o)

Input: A DARP instance $I = (T = (V, E), A, c, o)$.

Output: A tour for I, i. e., a closed walk containing all arcs from A.

1 Compute a balancing set B: $B :=$ BALANCE(T, A).
2 Compute the component graph H' for $(V, A \cup B)$.
3 Compute a minimum spanning tree S' of H'.
4 Construct a set of connecting arcs C: For each $\{u, v\} \in E$ used at least once by S' add arcs (u, v) and (v, u) to C.
5 Find an Euler tour $\mathcal{T}$ of $(V, A \cup B \cup C)$.
return $\mathcal{T}$

as defining a metric (V, D), where $D\colon V \times V \to \mathbb{R}_{\geq 0}$ is the lifted distance function of d.

Definition 2.10 (Star metric) A metric (M, d) is said to be a *star metric* if there is a $u^* \in M$ satisfying

$$d(u, v) = d(u, u^*) + d(u^*, v)$$

for all $u, v \in M$.

The proof of the following result is due to Sven O. Krumke.

Proposition 2.11 *Let $I = (G = (V, E), d\colon E \to \mathbb{R}_{\geq 0}, U \subseteq V)$ be an instance of* STEINERTREE. *Furthermore, let G' be the complete graph with vertex set U and denote by $D\colon U \times U \to \mathbb{R}_{\geq 0}$ the function assigning each pair (u_1, u_2) the length of a shortest path from u_1 to u_2 in G. Suppose that (U, D) is a star metric.*

We then have

$$\mathrm{MST}(G', D) = \textsc{SteinerTree}(G, d, U).$$

Proof. Consider an optimal Steiner tree S of (G, d, U) and let u^* be the center of the star metric (G', D). For the moment, assume that $|U|$ is even. The Pairing Lemma (see Lemma A.4) tells us that for every tree T with an even number of marked vertices $U \subseteq V$ we can arrange the vertices of U in pairs (u_1, u_2) such that all the (u_1, u_2)-paths are edge-disjoint. Therefore we can pair our terminals U in the tree $T = S$ in that way. Consider the edge set $S' \subseteq S$ which is the union of all the edges used in those paths.

Based on S' we can construct a spanning tree T' of G' which is not more longer than S': For each pair (u_1, u_2) add edges $\{u_1, u^*\}$ and $\{u^*, u_2\}$ to T'. The length of

the path u_1, u^*, u_2 is at most the length of the path from u_1 to u_2 used by our pairing due to the star metric property. The fact that those paths are edge-disjoint ensures that there is no hidden synergy between two or more paths that could share an edge and which would subvert the last argument. Clearly, T' is a tree, even a star graph.

We have just seen that each optimal Steiner tree corresponds to a MST that is not longer. Obviously, the other direction also holds. This proves $\mathrm{MST}(G', D) = \textsc{SteinerTree}(G, d, U)$ in the case of even $|U|$.

It remains to discuss the case that there is an odd number of terminals. This can easily be dealt with: Just copy a terminal and link the copy at distance 0 to its original. The copy shares all relevant properties of the original and we have thus reduced this case to that of an even number of terminals. □

The relevance of Proposition 2.11 in our context is that H' is just the graph U' for $G = H$. It follows that if (H', D_U) is a star metric, then USE-MST computes an optimal tour.

3 Probability Theory Basics for Probabilistic Analysis

This chapter is devoted to reviewing some basic concepts and results from Probability Theory. The presentation will be rather condensed and tailored to our purposes, but we try to elaborate on the way notions and results are used in probabilistic analyses. Basically we follow the overview given in the book of Motwani and Raghavan [29] and those of Janson et al. [25], which are also the recommended references for details on how to apply Probability Theory to the analysis of algorithms and of random combinatorial objects, respectively.

Probabilistic analysis of algorithms deals with the following situation: We have a (usually deterministic) algorithm which is run on an input generated by some random process. We are interested in the "typical" (instead of worst-case) performance of the algorithm, measured for instance by the running time or the solution quality. The "typical" performance is the performance on a large part of the instances where the "size" of a subset of the instances is measured by its probability. Thus we need to determine properties which govern the behavior of the algorithm and are enjoyed by most instances.

Before introducing tools and techniques for this sort of analysis we need some technical background.

3.1 Basic Notions: Random Objects and Random Variables

In this thesis we are interested in the properties of certain combinatorial objects (i. e., the input instances for our algorithms) constructed by some random process or random experiment. One usually models the outcomes of such a random process by a set Ω, called *sample space*. Our goal is to associate with every subset a probability. Once we have done this we have completely described our random experiment.

For technical reasons it is in general not possible to associate a probability to every subset of Ω in a consistent way. We have to restrict ourselves to a collection of subsets of Ω which has to carry a certain algebraic structure in order to facilitate the usual computations with such subsets, namely set-theoretic union, intersection and complement.

Definition 3.1 (σ-field) A *σ-field* is a pair $(\Omega, \mathcal{A})$ consisting of a sample space Ω and a collection of subsets of Ω satisfying the following axioms

1. $\emptyset \in \mathcal{A}$
2. $A \in \mathcal{A} \implies \overline{A} \in \mathcal{A}$
   ($\overline{A}$ denotes the complement of an event A w. r. t. Ω, i. e., $\overline{A} := \Omega \setminus A$.)
3. If countably many events $A_1, A_2, \ldots$ are in $\mathcal{A}$ then their union is also in $\mathcal{A}$: $A_1, A_2, \cdots \in \mathcal{A} \implies \bigcup_{i \in \mathbb{N}} A_i \in \mathcal{A}$.

An element of $\mathcal{A}$ called *event*.

The concept of σ-field allows us to define a function assigning probabilities to each event in $\mathcal{A}$. Of course, this function has to comply with some natural requirements.

Definition 3.2 (Probability measure) Let $(\Omega, \mathcal{A})$ be a σ-field. A *probability measure* is a function $\text{Prob}\colon \mathcal{A} \to [0, 1]$ with the properties

1. $\text{Prob}\,[\Omega] = 1$.
2. Suppose $A_1, A_2, \ldots$ are disjoint events. We then have
$$\text{Prob}\left[\bigcup\nolimits_{i \in \mathbb{N}} A_i\right] = \sum\nolimits_{i \in \mathbb{N}} \text{Prob}\,[A_i]\,.$$

Definition 3.3 (Probability space) A *probability space* is a triple $(\Omega, \mathcal{A}, \text{Prob})$ consisting of a σ-field $(\Omega, \mathcal{A})$ and a probability measure Prob for $(\Omega, \mathcal{A})$.

Note that a probability space is just the abstract description of a random experiment mentioned above. Recall that our goal was to construct random objects from a random experiment. So far we have only a mechanism for "throwing a dice" or "tossing a coin" but no way to derive something interesting from the outcomes.

Definition 3.4 (Random object, random variable) Let S be a countable set and Ω a countable sample space. A function $X\colon \Omega \to S$ is called a *random object* (or *random element of S*).

Consider a σ-field $(\Omega, \mathcal{A})$. A function $X\colon \Omega \to \mathbb{R}$ is called a *random variable* if for all $x \in \mathbb{R}$ the condition

$$\{\omega \in \Omega \mid X(\omega) \leq x\} \in \mathcal{A}$$

holds.*

Intuitively, the function $X\colon \Omega \to S$ tells us how to determine a fixed element of S from a specific outcome of a random experiment.

Example (Binomial model for random graphs) Suppose S is the set of graphs on n vertices and $\Omega = \{0,1\}^{\binom{n}{2}}$. We can construct a graph $G(x)$ from $x \in \Omega$ by fixing a numbering of all $\binom{n}{2}$ possible edges of a graph on n vertices and interpreting x_i as an indicator whether $G(x)$ contains edge i ($x_i = 1$) or not ($x_i = 0$). This is just the construction step and we still need to specify how elements of Ω are selected at random. In the binomial model, each component x_i is chosen to be one with probability p and to be zero with probability $1 - p$. Since there are only finitely many graphs on n vertices we can safely choose $\mathcal{A} = 2^\Omega$ and the probability measure consistent with the above interpretation is

$$\text{Prob}\,[G(x) = G] = p^{m(G)}(1-p)^{\binom{n}{2}-m(G)},$$

where $m(G)$ denotes the number of edges of G. ∎

Before investigating random objects in more detail we introduce two further concepts. Suppose we are considering a fixed random experiment described by the probability space $(\Omega, \mathcal{A}, \text{Prob})$ and we already know that some event B has occurred and now want to examine how this influences the probability of another event A. Clearly, the probability of A must be 0 if $A \cap B = \emptyset$, i. e., if A and B are mutually exclusive. In general only the fraction of A compatible with B does contribute to the probability of A.

Definition 3.5 (Conditional probability) Fix a probability space $(\Omega, \mathcal{A}, \text{Prob})$ and an event $B \in \mathcal{A}$ with $\text{Prob}\,[B] > 0$. For any event $A \in \mathcal{A}$, the probability

$$\text{Prob}\,[A \mid B] := \frac{\text{Prob}\,[A \cap B]}{\text{Prob}\,[B]}$$

is called *conditional probability of A given B*. The function $\text{Prob}\,[\cdot \mid B]$ is a probability measure on $(\Omega, \mathcal{A})$.

*A similar *measurability* condition would be necessary for the case of a countable set S, too. However, in that case the function X is always measurable w. r. t. the sigma field $(\Omega, 2^\Omega)$, so the condition is satisfied automatically.

Conditional probabilities are useful because in many circumstances it is easy to compute the probability of some event A conditional on another event B. If the probability of B can also be determined we can use it to obtain the probability of both A and B by

$$\text{Prob}\left[A \cap B\right] = \text{Prob}\left[A \mid B\right] \text{Prob}\left[B\right].$$

It is often advantageous to deal with random objects (variables) that "have nothing to do with and do not influence each other".

Definition 3.6 (Independence) Let $(\Omega, \mathcal{A}, \text{Prob})$ be a probability space and consider random objects $X_i \colon \Omega \to S_i$, $1 \leq i \leq k$. The X_i are called *independent* if

$$\text{Prob}\left[X_1 = s_1, \ldots, X_k = s_k\right] = \text{Prob}\left[X_1 = s_1\right] \cdots \text{Prob}\left[X_k = s_k\right]$$

for all $s_i \in S_i$, $1 \leq i \leq k$. Similarly, random variables $Y_i \colon \Omega \to \mathbb{R}$, $1 \leq i \leq k$, are said to be *independent* if

$$\text{Prob}\left[Y_1 \leq y_1, \ldots, Y_k \leq y_k\right] = \text{Prob}\left[Y_1 \leq y_1\right] \cdots \text{Prob}\left[Y_k \leq y_k\right]$$

for all combinations $y_1, \ldots, y_k \in \mathbb{R}$.

The intuition that two independent random objects X_1, X_2 "have nothing to do with each other" can be illustrated as follows: The conditional probability of $\{X_1 = s_1\}$ given $\{X_2 = s_2\}$ is

$$\begin{aligned}\text{Prob}\left[X_1 = s_1 \mid X_2 = s_2\right] &= \frac{\text{Prob}\left[X_1 = s_1 \cap X_2 = s_2\right]}{\text{Prob}\left[X_2 = s_2\right]} \\ &= \frac{\text{Prob}\left[X_1 = s_1\right] \text{Prob}\left[X_2 = s_2\right]}{\text{Prob}\left[X_2 = s_2\right]} = \text{Prob}\left[X_1 = s_1\right].\end{aligned}$$

According to our interpretation of conditional probabilities the fact that $X_2 = s_2$ does not tell us anything about X_1.

The standard construction for independent random objects is to use product probability spaces: Suppose we are given probability spaces $(\Omega_i, \mathcal{A}_i, \text{Prob}_i)$ and random objects (or random variables) $X_i \colon \Omega_i \to S_i$, $1 \leq i \leq k$. So far these random objects are entirely unrelated. In order to relate them to each other we use the *product probability space* $(\Omega, \mathcal{A}, \text{Prob})$ defined as

$$\begin{aligned}\Omega &:= \Omega_1 \times \cdots \times \Omega_k \\ \mathcal{A} &:= \mathcal{A}_1 \times \cdots \times \mathcal{A}_k \\ \text{Prob}\left[(A_1, \ldots, A_k)\right] &:= \text{Prob}_1\left[A_1\right] \cdots \text{Prob}_k\left[A_k\right] \quad \forall (A_1, \ldots, A_k) \in \mathcal{A}.\end{aligned}$$

If we think of the random object X_i as being defined on the ith component of Ω, we see that the X_i are independent by construction, since the conditions $X_i = s_i$ (or $Y_i \leq y_i$) used in the definition of independence are just special events of $\mathcal{A}$. Of course this construction is often kept implicit.

Now that we know what random objects are we turn to study their properties. There are two kinds of properties: Numerical ones (for instance the number of edges of a graph, the number of connected components, the length of a shortest path between two vertices) and non-numerical ones (Is the graph connected? Does it have a cycle? Is it Eulerian?). We will first address numerical properties.

3.2 Numerical Properties of Random Objects

Note that any numerical value associated with a random object can be considered as a random variable X: First, construct the random object and then compute the numerical quantity from it. The majority of all random variables occurring in this thesis are of this type. One way to specify a random variable is by giving its distribution function.

Definition 3.7 (Distribution function) Let $X\colon \Omega \to \mathbb{R}$ be a random variable. The *distribution function* $F_X\colon \mathbb{R} \to [0,1]$ *of* X is defined by

$$F_X(x) := \text{Prob}\,[X \leq x]\,.$$

For technical reasons we discriminate two types of random variables.

Definition 3.8 (Discrete vs. continuous random variables) Let $X\colon \Omega \to \mathbb{R}$ be a random variable. If the range of X is a countable subset $S \subset \mathbb{R}$, X is a *discrete random variable*.

A random variable is called *continuous* if there is a non-negative function $f_X\colon \mathbb{R} \to \mathbb{R}_{\geq 0}$ such that

$$F_X(x) = \int_{t=-\infty}^{x} f_X(t)\ \mathrm{d}t. \tag{3.1}$$

The function f_X is called *density of* X.

For every discrete random variable X there is a discrete analogue of the density: One often writes it explicitly as $\text{Prob}\,[X = x] = p_x$ for all possible values x. The

distribution function is completely determined by these probabilities similarly to Equation (3.1):

$$F_X(x) = \sum\nolimits_{t \in S, t \leq x} \text{Prob}\left[X = t\right].$$

Since it is often difficult to cope with random variables in general, we are interested in some key characteristics of them.

Definition 3.9 (Expectation, Variance) Let X be a random variable. The *expectation* $\mathbb{E}\left[X\right]$ *of* X is defined by

$$\mathbb{E}\left[X\right] := \begin{cases} \sum_{t \in S} t \cdot \text{Prob}\left[X = t\right] & X \text{ is discrete} \\ \int_{t=-\infty}^{\infty} t \cdot f_X(t)\ \mathrm{d}t & X \text{ is continuous} \end{cases} \tag{3.2}$$

whenever the right hand side is $< \infty$.

The *variance* $\text{Var}\left[X\right]$ *of* X is

$$\text{Var}\left[X\right] := \mathbb{E}\left[(X - \mathbb{E}\left[X\right])^2\right]. \tag{3.3}$$

The expectation of a random variable is its average value, whereas the variance measures the "typical deviation" from the average: The larger the variance, the more likely are values relatively far away from the expectation.

Example (Important probability distributions) A simple discrete probability distribution is the *uniform distribution on* $S = \{x_1, \ldots, x_k\}$ given by

$$\text{Prob}\left[X = x\right] = \frac{1}{k} \quad \forall x \in S.$$

Another well-known discrete distribution is the *Poisson distribution with parameter* $\lambda > 0$

$$\text{Prob}\left[X = k\right] = \frac{\lambda^k}{k!}\mathrm{e}^{-\lambda} \quad \forall k \in \mathbb{N}_0.$$

If X is Poisson-distributed we have that $\mathbb{E}\left[X\right] = \text{Var}\left[X\right] = \lambda$.

The *exponential distribution with parameter* $\lambda > 0$ is a continuous distribution and defined by

$$F_X(t) = \begin{cases} 0 & x < 0 \\ 1 - \mathrm{e}^{-\lambda t} & x \geq 0 \end{cases}$$

and it is easy to see that $\mathbb{E}\left[X\right] = \frac{1}{\lambda}$ and $\text{Var}\left[X\right] = \frac{1}{\lambda^2}$. ■

The following rules greatly simplify dealing with expectations and variances.

Theorem 3.10 *Let $X_1, \ldots, X_n$ be random variables.*

1. *For any reals $a_i, b_i \in \mathbb{R}$, $1 \leq i \leq n$, we have the well-known linearity of expectation:*

$$\mathbb{E}\left[\sum_{i=1}^{n}(a_i X_i + b_i)\right] = \sum_{i=1}^{n}(a_i \mathbb{E}[X_i] + b_i). \tag{3.4}$$

2. $\operatorname{Var}[X_1] = \mathbb{E}[X_1^2] - \mathbb{E}[X_1]^2$.

3. *If $X_1, \ldots, X_n$ are independent random variables we have*

$$\operatorname{Var}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \operatorname{Var}[X_i]. \qquad \square$$

In applications it is often crucial to know that a random variable is in a certain range with a high probability. Many random variables are well-behaved in this respect and do not deviate far from their expectation. To prove such statements a large class of *tail inequalities* or *concentration of measure inequalities* have been developed. The simplest of them is *Markov's inequality* stating that

$$\operatorname{Prob}[X \geq t] \leq \frac{\mathbb{E}[X]}{t} \tag{3.5}$$

for any non-negative random variable X. We can see that it is unlikely that X is large provided that $\mathbb{E}[X]$ is small. Another related one is *Chebyshev's inequality*

$$\operatorname{Prob}[|X - \mathbb{E}[X]| \geq t] \leq \frac{\operatorname{Var}[X]}{t^2}, \tag{3.6}$$

which is valid for any random variable X. It is more useful if the expectation is large.

The main use of expectations and variances in this thesis will be to arrive at such results. However, both the Markov and Chebyshev inequalities are sometimes to weak and we need stronger results, for example the following theorem which is related to the so-called *Azuma's inequality*. For a proof and discussion see [25].

Theorem 3.11 *Let $X_1, \ldots, X_n$ be independent random variables $X_i \colon \Omega \to \mathbb{R}$, $1 \leq i \leq n$, and $f \colon \mathbb{R}^n \to \mathbb{R}$ a function satisfying the Lipschitz-condition*

Each pair of vectors $x, x' \in S_1 \times \cdots \times S_n$ differing only in the kth coordinate satisfies

$$|f(x) - f(x')| \leq c_k \tag{3.7}$$

for some suitable constants $c_i \in \mathbb{R}_{\geq 0}, 1 \leq i \leq n$.

Then, the random variable $Y = f(X_1, \ldots, X_n)$ satisfies for each $t \geq 0$

$$\text{Prob}\left[Y \geq \mathbb{E}\left[Y\right] + t\right] \leq \exp\left(-\frac{t^2}{2\sum_1^n c_k^2}\right) \tag{3.8}$$

$$\text{Prob}\left[Y \leq \mathbb{E}\left[Y\right] - t\right] \leq \exp\left(-\frac{t^2}{2\sum_1^n c_k^2}\right). \tag{3.9}$$

□

3.3 Non-numerical Properties of Random Objects

Let us now consider non-numerical properties. The motivating examples where all questions such as "Does a graph have a cycle?" which can be answered for each concrete object by yes or no. This suggests formalizing such properties by a predicate function ϕ which assigns to each object enjoying the property under consideration a "1" and a "0" otherwise. Remember that our random objects are by definition constructed from events of the sample space. Thus we can as well characterize our property by all events leading to random objects satisfying this property.

Definition 3.12 (Asymptotic properties) We say that a property defined by a family of events $\{A_n\}_{n\in\mathbb{N}}$ holds *asymptotically almost surely* (*a. a. s.* for short), if

$$\text{Prob}\left[A_n\right] = 1 - o(1) \quad \text{as } n \to \infty. \tag{3.10}$$

To make the above idea a bit more explicit, let $\{X_n \colon \mathcal{A}_n \to S_n\}_{n\in\mathbb{N}}$ be a family of random objects indexed by a parameter n which is interpreted as the size of the random object. The random objects are defined on a family of probability spaces $\{(\Omega_n, \mathcal{A}_n, \text{Prob}_n)\}_{n\in\mathbb{N}}$. Moreover, let ϕ be a predicate as explained above. If the family of events

$$A_n := \{\omega \in \Omega_n \mid \phi\big(X_n(\omega)\big) = 1\}$$

satisfies condition (3.10) this means that as the objects X_n get larger and larger, a growing fraction of them has property ϕ. Intuitively, almost all of the really large objects exhibit property ϕ, which is thus in a sense typical.

Often it is not easy to show directly that property ϕ is satisfied a. a. s. We can help ourselves by finding properties $\phi_1, \ldots, \phi_k$ such that

$$\phi_1 \wedge \cdots \wedge \phi_k \implies \phi$$

and showing that $\phi_1, \ldots, \phi_k$ are satisfied a. a. s., allowing us to conclude that ϕ is also satisfied a. a. s. This is justified by the following computation: Assume that A_n

and B_n are the event sets corresponding to properties ϕ_A and ϕ_B, respectively, which hold a. a. s. The probability that $\phi \Longleftarrow \phi_A \wedge \phi_B$ holds is at least

$$\begin{aligned}\text{Prob}\left[A_n \cap B_n\right] &= 1 - \text{Prob}\left[\overline{A_n \cap B_n}\right] \\ &= 1 - \text{Prob}\left[\overline{A_n} \cup \overline{B_n}\right] \\ &= 1 - \text{Prob}\left[\overline{A_n}\right] - \text{Prob}\left[\overline{B_n}\right] + \text{Prob}\left[\overline{A_n} \cap \overline{B_n}\right] \\ &\geq 1 - o(1) - o(1) = 1 - o(1) \quad \text{as } n \to \infty.\end{aligned}$$

Note that finding suitable properties $\phi_1, \ldots, \phi_k$ is a purely deterministic consideration, so all the deterministic theory on the objects in question can be employed.

We often can resort to numerical properties, such as the expectation, to prove that a certain property is satisfied a. a. s. The main advantage is that they can be better handled and there is a whole arsenal of methods to deal with them. An important example are the *first and second moment methods*. Suppose the family of random variables $\{X_n \colon \mathcal{A}_n \to \mathbb{R}\}_{n\in\mathbb{N}}$ counts the number of a "bad" substructure of a random object (such as the small loops in a graph that is supposed to be a big loop) and we want to show that a. a. s. there are no "bad" substructures, i. e., $X_n = 0$ a. a. s. We can do this by establishing $\mathbb{E}[X_n] = o(1)$ as $n \to \infty$ since then Markov's inequality (3.5) says that

$$\text{Prob}[X_n > 0] = \text{Prob}[X_n \geq 1] \leq \mathbb{E}[X] = o(1).$$

This kind of argument is valid for any non-negative integer-valued family of random variables and is called *first moment method*.

In other cases we want to have a. a. s.-lower or upper bounds on X_n. We can show that X_n is within constant factors of $\mathbb{E}[X_n]$ a. a. s. if

$$\frac{\text{Var}[X]}{\mathbb{E}[X]^2} \in o(1) \quad \text{as } n \to \infty$$

because Chebyshev's inequality yields

$$\text{Prob}\left[|X - \mathbb{E}[X]| \geq c\mathbb{E}[X]\right] \leq \frac{\text{Var}[X]}{c^2\mathbb{E}[X]^2} \in o(1) \quad \text{as } n \to \infty.$$

This is known as the *second moment method*.

Yet another useful technique for analyzing random objects is to consider a different random experiment for generating them, which is better adapted to the analysis applied. Of course one has to ensure that the new random model is in some way equivalent to the original one which means that the probabilities that a fixed object is constructed do agree.

4 Probabilistic Analysis of the DARP on Trees

The investigations of this chapter are motivated by the following, seemingly contradictory, facts: On the one hand DARP is known to be NP-hard on trees, on the other hand it has been observed experimentally that the algorithm USE-MST introduced in Chapter 2 solves many instances of DARP optimally, although its approximation ratio is only $\frac{4}{3}$ in the worst case. This suggests that instances exhibiting bad or even worst-case behaviour of USE-MST are rare.

To substantiate this conjecture Coja-Oghlan et al. [12] performed a probabilistic analysis of USE-MST on random instances which is reviewed in this chapter. Recall that the only situation in which USE-MST does not find am optimal solution is when the minimum spanning tree in the component graph is more expensive than a Steiner tree of the arc-identified graph, i. e., if the MST-heuristic for STEINERTREE fails. Thus most instances have to feature some structure implying that the MST-heuristic returns a tree equivalent to an optimal Steiner tree. One would first hypothesise that the balancing operation leaves only one strongly connected component but it turns out that this is not sufficient. The correct generalisation is that the components left by the balancing operation make up a star metric.

Before delving into the technical details of this result we first give an overview and a brief discussion. The second section explains the structure of the proof of the main theorem from a higher perspective but with technical details. In the third section we provide formal statements and proofs only motivated in Section 4.2. The final Sections 4.4 to 4.6 present intermediate results. The proof of the main theorem is in Section 4.6, too.

4.1 Overview and Key Ideas

At the heart of this analysis is the following probabilistic model for the DARP instances. The key feature of this model is that the underlying network and the cor-

responding distance function are assumed to be fixed and the requests are generated according to a probability distribution. The reason for this separation is that in practice the topology and the transportation cost of the transportation networks are fixed. In contrast, there are lots of requests which are in principle unpredictable but may exhibit some statistical structure.

Definition 4.1 (List random model) Let the tree $T = (V, E)$, the depot vertex $o \in V$ and the distance function $d\colon E \to \mathbb{R}_{\geq 0}$ be fixed and assume that $(p_v)_{v \in V}$ is a probability distribution on the vertex set V. Then a *random request list* L_m *of length* m is constructed by randomly choosing m pairs of source and destination vertices according to $(p_v)_{v \in V}$. Denote by $A(L_m)$ the multiset of arcs corresponding to the request list L_m. The random DARP instance is now $I = (T, A(L_m), d, o)$.

Note that the list random model includes the important uniform distribution on the vertices (leading to the uniform request distribution on $V \times V$) as a special case.

The main result of the probabilistic analysis is summarized by the following theorem.

Theorem 4.2 *Let $I = (T, A, c, o)$ be a random* DARP *instance where the request set $A = A(L_m)$ is chosen randomly according to the list random model. The algorithm* USE-MST *(see Algorithm 2.4) solves I optimally a. a. s., that is*

$$\text{Prob}\left[\text{USE-MST}(I) = \text{DARP}(I)\right] = 1 - o(1) \quad \text{as } m \to \infty.$$

Moreover, if the tour output by USE-MST *is optimal there is a. a. s. a certificate of optimality which can be computed in polynomial time.*

This result follows from the fact that the graph $(V, A \cup B)$ arising from the balancing operation is a. a. s. such that its component graph forms a star metric. We know that then the result of the MST-heuristic corresponds to an optimal solution of the STEINERTREE-Problem for the arc-identified graph (see Proposition 2.11). Furthermore it can be checked in polynomial time whether the component graph is a star metric or not, thereby providing the promised certificate.

Some remarks on the result of Theorem 4.2 are in order. First of all, it is an asymptotic result. That means that it does not tell us much about "small" instances – it may well be that this result does not give us anything for all real-world-size instances. Indeed the probability that "everything goes well" increases to 1 rather slowly. On the other hand we know empirically from experiments that USE-MST is well-behaved on small instances so we may look at both results as complementing each other.

Another issue we want to emphasise is that this result is (asymptotically) much stronger than any *average case result* on USE-MST: The algorithm USE-MST has always polynomial running time (in fact, nearly-linear running time) and is optimal on almost all large instances, whereas an algorithm with good performance on average may be bad on a large part of the instances.

The main idea for this probabilistic analysis is to use a reformulation of the list random model featuring more stochastic independence. We then proceed by the following steps:

1. We first show that the *balancing arcs alone* a. a. s. make up less than one component per request, i. e., there are a. a. s. less than m components in the graph (V, B) (Section 4.4).

2. Next we consider what happens to these balancing arcs components if we add the request arcs. It turns out that there are a. a. s. one large nontrivial component and some small ones (Section 4.5).

3. The last step consists of showing that a. a. s. each path starting in one of the small nontrivial components and ending in another nontrivial component has to pass through the large nontrivial component (Section 4.6). This ensures that we have a star metric on the component graph and thus concludes the proof of Theorem 4.2.

4.2 A More Technical Road Map

We now sketch the ideas underlying the reformulation of the list random model. At the end of this section we discuss how this reformulation helps us in the analysis.

We use the same notation as in Chapter 2 although all the symbols depending on the DARP instance are now random objects or random variables. Recall that an edge $\{u, v\}$ is used by the balancing set B output by BALANCE (see Algorithm 2.1) if and only if $b(u, v) + b(v, u) > 0$, where

$$b(u,v) = \begin{cases} 1 & \Phi(u,v) = \Phi(v,u) = 0 \\ \Phi(v,u) - \Phi(u,v) & \Phi(v,u) > \Phi(u,v) \\ 0 & \text{else.} \end{cases} \tag{2.2}$$

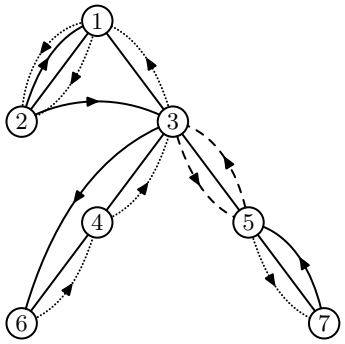


Figure 4.1: The balancing set for the DARP instance of Figure 2.1(a) divided into arc sets B' (dotted) and B'' (dashed).

The balancing arcs in B can be partitioned into arc multisets B' and B'' (see Figure 4.1)

$$B' := \{(u,v) \mid \Phi(v,u) > \Phi(u,v)\}$$
$$B'' := \{(u,v) \mid \Phi(u,v) = \Phi(v,u) = 0\}.$$

We saw in the justification of Equation (2.2) that the arcs in B' augment the request set A to form strongly connected Eulerian components. In contrast, arcs from B'' connect some of those components to larger ones. The analysis presented here deals only with arcs in B'. By definition, B' is a submultiset of B, so every component of $(V, A \cup B)$ corresponds to one or more components of $(V, A \cup B')$.

This restricted balancing operation requires the computation of $\Phi(v,u) - \Phi(u,v)$, which can conveniently be expressed by using the in- and outdegrees of $T_A := (V, A)$:

$$\begin{aligned}
\Phi(v,u) - \Phi(u,v) &= |V(v) \times V(u) \cap A| - |V(u) \times V(v) \cap A| \\
&= |V(v) \times V \cap A| - |V(v) \times V(v) \cap A| \\
&\quad - \big(|V \times V(v) \cap A| - |V(v) \times V(v) \cap A|\big) \\
&= |V(v) \times V \cap A| - |V \times V(v) \cap A| \\
&= \delta^+_{T_A}(V(v)) - \delta^-_{T_A}(V(v)). \qquad (4.1)
\end{aligned}$$

Intuitively, the surplus of requests traversing $\{u,v\}$ from v to u over those in the opposite direction is just the number of requests starting in $V(v)$ minus the number of requests ending in $V(v)$.

Notice that $\Phi(v,u) - \Phi(u,v)$ depends only on the in- and outdegrees of T_A, that is, for a request starting in v the destination vertex is irrelevant. We have thus

decoupled the in- and outdegree information from the actual connection information, which will be exploited by the modified list random model.

Modified list random model We define a modified random model which is not fully equivalent to the standard list random model (see Definition 4.1). It is only equivalent in a restricted sense: Consider all random request lists L_m having a certain fixed request-vertex-incidence vector $\chi = (\chi_v)_{v \in V}$. Our modified list random model generates request sets according to χ with the same distribution as the standard list random model conditioned on $\chi(L_m) = \chi$. This will be made more precise in Proposition 4.7.

The philosophy behind this restriction is the following: For each fixed χ we can imagine random instances as being generated by the modified list random model, which facilitates our analysis. We show that for each χ the (restricted) balancing of random instances results in a star metric, so this property holds in general.

The request-vertex-incidence vector χ is where our probability distribution $(p_v)_{v \in V}$ comes into play: We may think of χ as being generated by choosing $2m$ vertices according to $(p_v)_{v \in V}$; χ_v is then just the number of times v has been chosen. The vector χ can be interpreted as assigning a vertex v exactly χ_v "slots", which may be used as either source or destination of a request. As source and destination vertex are chosen according to the same probability distribution, we can first decide on the number of requests incident to a vertex v (namely χ_v) and in a second step choose whether a "slot" will be a source or destination such that there are m source and destination "slots". It then remains to connect them. To simplify the random experiment for achieving this and because we need a more structured T in the subsequent analysis, we will assume a normalized T.

More precisely, we will assume that both the source and the destination vertex of a request are leaves and that each leaf is either source or destination of *exactly one* request thus the leaves will play the role of the "slots" mentioned above. Furthermore we need the technical requirement that T is a rooted *full binary tree*, i. e., each non-leaf has exactly two children. It is possible to transform an arbitrary tree to meet this requirements such that if the component graph of the transformed tree is a star metric so is the component graph of the original tree. The transformation depends only on T and the fixed request-vertex-incidence vector χ and is described later. Its properties are given by Lemma 4.5 and the preservation of the star metric is the statement of Proposition 4.6.

Note that this transformation is a purely technical device to simplify the proof of

Theorem 4.2, since it justifies that we can restrict ourselves to full binary trees with requests starting and ending at leaves. This transformation is *not* carried out for the original instance but only used to show that the original instance gives rise to a star metric.

Henceforth we will assume that the tree T and all related items have been normalized. We are now able to describe the modified list random model.

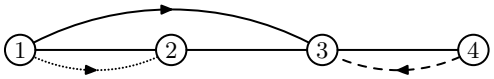
Definition 4.3 (Modified list random model) Let T be a full binary tree with exactly $2m$ leaves. A random request list is now constructed as follows:

1. Choose a vector $x \in \{+1, -1\}^{2m}$ such that $\sum_{1 \le i \le 2m} x_i = 0$ at random. This is interpreted as "leaf i is source of a request" ($x_i = +1$) or "leaf i is destination of a request" ($x_i = -1$).

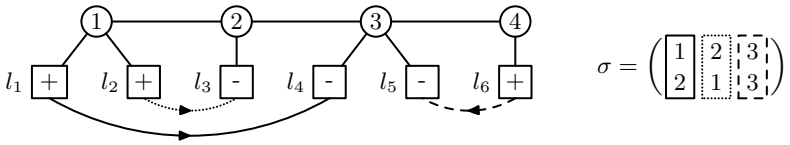
2. Choose a permutation $\sigma\colon \{1, \dots, m\} \to \{1, \dots, m\}$ at random. The permutation σ determines how to connect the "+1"-leaves to "−1"-leaves: The ith leaf having $x_i = +1$ is connected to the $\sigma(i)$th leaf having $x_{\sigma(i)} = -1$.

We get a random request set $A(x, \sigma)$ this way.

Example Suppose the list random model produced a request set $A(L_3) = \big\{(1,3), (1,2), (4,3)\big\}$, depicted in the following figure.



An equivalent request set would be obtained by the modified list random model for $x = (+1, +1, -1, -1, -1, +1)$ and $\sigma = \left(\begin{smallmatrix}1&2&3\\2&1&3\end{smallmatrix}\right)$. These parameters yield the request set $A(x, \sigma) = \big\{(l_1, l_4), (l_2, l_3), (l_6, l_5)\big\}$:



Note that we did not insist on the full binary tree requirement for the purposes of this example. ∎

We may choose whether a leaf is source or destination with equal probability *after* fixing the request-vertex-incidence vector χ because the list random model uses the *same* distribution $(p_v)_{v \in V}$ for both source and destination of a request.

As mentioned before the modified list random model is equivalent to the list random model conditioned on a fixed request-vertex-incidence vector χ. This is formally stated in Proposition 4.7.

Road map of the analysis In order to pursue the program stated at the end of the last section we need to describe both the set of balancing arcs B' and the request set A in terms of the modified list random model. Since we know that any weakly connected component of $(V, A \cup B')$ is indeed strongly connected (see Lemma 2.3) it suffices to analyse weakly connected components and we can restrict ourselves to edge sets (instead of arc sets). We can describe the underlying edge sets of A and B' by exploiting the modified list random model as follows:

$$E_{B'}(x) = \big\{\{u, v\} \mid \delta^+_{T_A}(V(v)) - \delta^-_{T_A}(V(v)) \neq 0\big\}$$
$$E_A(x, \sigma) = \big\{\{u, v\} \mid (u, v) \in A(x, \sigma)\big\}.$$

Our tree T is rooted w. r. t. to the root vertex r, allowing us to express the set $V(v)$ induced by an edge $\{v, \varrho(v)\}$ as $V(v) = \{v' \in V \mid v' \preceq v\}$, where $\varrho(v)$ denotes the parent vertex of a vertex v. The partial order on the vertices used here is defined as

$$u \preceq v \iff v \text{ is on the unique path from } u \text{ to the root } r.$$

We also say that v *majorizes* u.

The analysis now proceeds as follows:

- First we show that $T_{B'} := (V, E_{B'})$ has few components a. a. s. To this end, we define the random variable $S_v(x)$ indicating the "balance" of a vertex v:

$$S_v(x) := \sum\nolimits_{v' \preceq v} (\delta^+_{T_A}(v') - \delta^-_{T_A}(v'))$$
$$= \sum\nolimits_{l_i \preceq v} x_i,$$

  so we have

$$E_{B'}(x) = \big\{\{v, \varrho(v)\} \mid S_v(x) \neq 0, v \neq r\big\}.$$

  If v is the maximum vertex of a component of $T_{B'}$ we have $S_v(x) = 0$, so we can use the number of vertices satisfying $S_v(x) = 0$ as an estimate for the number of components of $T_{B'}$.

- Second we consider what happens if $T_{B'}$ is augmented with the request edges $E_A(x, \sigma)$ to obtain $T_{AB'} := (V, E_{B'} \cup E_A)$: As it turns out, there is a. a. s. one component C^* containing more than half of the leaves and all other components are a. a. s. cycles.

- The last step consists of showing that the path from vertex u to vertex v from the remaining components crosses C^* a. a. s., which establishes the star metric property.

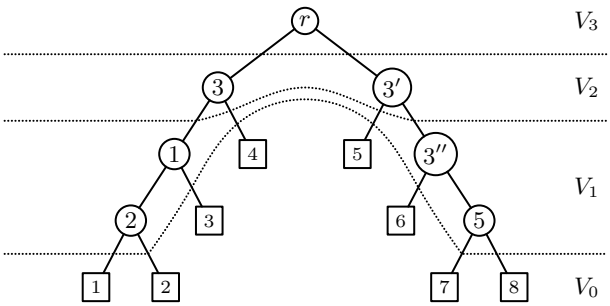


Figure 4.2: Layers in the vertex set of the normalized version (cf. Figure 4.3) of the DARP instance shown in Figures 4.1 and 2.1(a).

We will need a little more notation. Let $l(v)$ be the number of leaves below a vertex v. We partition the vertex set V in "layers" (see Figure 4.2), depending on how many leaves each vertex majorizes:

$$V_j := \{v \in V \mid 2^j \leq l(v) < 2^{j+1}\}, \qquad 0 \leq j \leq \lceil \log_2(2m) \rceil \,. \tag{4.2}$$

Moreover, we set

$$V_{\leq k} := \bigcup\nolimits_{j \leq k} V_j \qquad \text{and} \qquad V_{\geq k} := \bigcup\nolimits_{j \geq k} V_j.$$

It will prove useful in the later discussion to know something about the number of maximal and minimal vertices of V_j. We can exploit the binary tree structure to obtain such bounds.

Lemma 4.4 *There are at most $\frac{2m}{2^j}$ maximal and minimum vertices w. r. t. $\preceq$ in V_j.*

Proof. Different maximal vertices $u, v \in V_j$ majorize disjoint sets of leaves. Since each maximal vertex of V_j majorizes at least 2^j leaves, there can be at most $\frac{2n}{2^j}$ distinct maximal vertices in V_j.

To conclude that the number of minimal vertices is also at most $\frac{2n}{2^j}$ we establish a one-to-one-correspondence between the maximal and minimal vertices of V_j. To this end, we show that the set

$$M(v) := \{u \in V_j \mid u \preceq v\}$$

of vertices in V_j majorized by a fixed maximal vertex $v \in V_j$ is a chain w. r. t. $\preceq$.

To construct a contradiction, assume that $u, u' \in M(v)$ are such that $u \not\preceq u'$ and $u' \not\preceq u$. Since both u, u' are elements of V_j, we know that $l(u), l(u') \geq 2^j$. Furthermore, u and u' majorize disjoint leaf sets, so we have $l(v) \geq l(u) + l(u') \geq 2^{j+1}$, contradicting $v \in V_j$. □

4.3 Some Preliminaries

In this section basic results used in the following sections are proven. Most of them have been mentioned and motivated in the preceding discussion.

First we have to verify our claim that a transformation with the properties described in the last section exists. Algorithm NormalizeTree is one way to do such a transformation and the properties of its output are summarized by the following lemma. Note that in order to establish the required properties we have to ensure that the distances of the vertices are the same. Transformed items are indicated by a bar.

Lemma 4.5 *The algorithm* NormalizeTree *(see Algorithm 4.1) transforms a tree $T = (V, E)$, its edge length function $d\colon E \to \mathbb{R}_{\geq 0}$ and the vector χ to another tree $\bar{T} = (\bar{V}, \bar{E})$, associated edge length function $\bar{d}\colon \bar{E} \to \mathbb{R}_{\geq 0}$ and vector $\bar{\chi}$, which are equivalent for our purposes. More specifically, $\bar{T}$, $\bar{d}$ and $\bar{\chi}$ satisfy*

1. *$\bar{T}$ is a full binary tree: each non-leaf has exactly two children*
2. *Each "slot" is a leaf and vice versa: $\bar{\chi}_v = 0$ for all non-leaves v and $\bar{\chi}_{l_i} = 1$ for all leaves l_i. Furthermore, there are χ_v leaves for vertex $v \in V$, which are at distance 0 to a corresponding vertex for v in $\bar{T}$.*
3. *The distances do not change: If $\bar{u}, \bar{v}$ are corresponding vertices of $u, v \in V$, we have $\bar{D}(\bar{u}, \bar{v}) = D(u, v)$. (Recall that D denotes the lifted distance function.)*
4. *In both subtrees of the root vertex r are at least a third of the leaves.*

Proof. By inspection of the algorithm (see Figure 4.3 for an example). □

The purpose of the transformation process is to justify the restriction to full binary trees with requests starting and ending at leaves only. The following proposition gives us this justification.

Proposition 4.6 *Let $T = (V, E)$ be an arbitrary tree with associated edge length function $d\colon E \to \mathbb{R}_{\geq 0}$, χ a request-vertex-incidence vector, L_m a list of requests with $\chi(L_m) = \chi$ and finally B' the arc multiset resulting from the restricted balancing process on T and $A(L_m)$. Furthermore, let $\bar{T}$, $\bar{d}$, $\bar{\chi}$ and $\bar{L}_m$ be the corresponding items as transformed by* NormalizeTree *and $\bar{B}'$ the result from restricted balancing.*

If the component graph $\bar{H}'$ of $(\bar{V}, A(\bar{L}_m) \cup \bar{B}')$ is a star metric, so is the component graph H' of $(V, A(L_m) \cup B')$.

Algorithm 4.1 Algorithm NORMALIZETREE for transforming a tree T to a canonic structure.

NORMALIZETREE (T, d, χ)

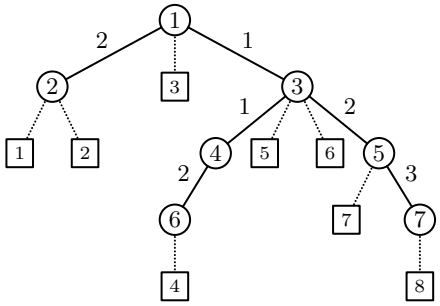
Input: Tree $T = (V, E)$, edge length function $d\colon E \to \mathbb{R}_{\geq 0}$, request-vertex-incidence vector $\chi = (\chi_v)_{v \in V}$.

Output: Modified tree $\bar{T} = (\bar{V}, \bar{E})$, edge length function $\bar{d}\colon \bar{E} \to \mathbb{R}_{\geq 0}$, request-vertex-incidence vector $\bar{\chi} = (\bar{\chi}_v)_{v \in \bar{V}}$.

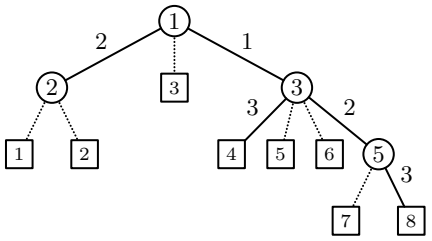
$\bar{V} := V; \bar{E} := E; \bar{d} := d; \bar{\chi} := \chi$
for $v \in \bar{V}$ **do**
  Create $\bar{\chi}_v$ leaves $l_1, \ldots, l_{\bar{\chi}_v}$ and connect them to v at distance 0. Set $\bar{\chi}_v = 0$ and $\bar{\chi}_{l_i} = 1$.
end for
for $v \in \bar{V}$ and $\delta_{\bar{T}}(v) = 2$ **do**
  Let $\{u, v\}$ and $\{v, w\}$ be the incident edges.
  Remove v from $\bar{V}$ and replace $\{u, v\}$ and $\{v, w\}$ by $\{u, w\}$ in $\bar{E}$.
  Set $\bar{d}(\{u, w\}) := \bar{d}(\{u, v\}) + \bar{d}(\{v, w\})$.
end for
for $v \in \bar{V}$ **do**
  while $\delta_{\bar{T}}(v) > 3$ **do**
    Let w_1, w_2 be neighbors of v.
    Remove edges $\{v, w_1\}$ and $\{v, w_2\}$ from $\bar{E}$.
    Add a new vertex v' to $\bar{V}$ and add edges $\{v, v'\}$, $\{v', w_1\}$ and $\{v', w_2\}$ to $\bar{E}$.
    Set $\bar{d}(\{v, v'\}) := 0$
      $\bar{d}(\{v', w_1\}) := \bar{d}(\{v, w_1\})$
      $\bar{d}(\{v', w_2\}) := \bar{d}(\{v, w_2\})$.
  end while
end for
Insert a root vertex r by splitting an edge $\{u, v\}$ such that $V(u)$ and $V(v)$ contain at least $2m/3$ leaves. The distances after splitting are

$$\bar{d}(\{u, r\}) := \tfrac{1}{2}\bar{d}(\{u, v\})$$
$$\bar{d}(\{r, v\}) := \tfrac{1}{2}\bar{d}(\{u, v\}).$$

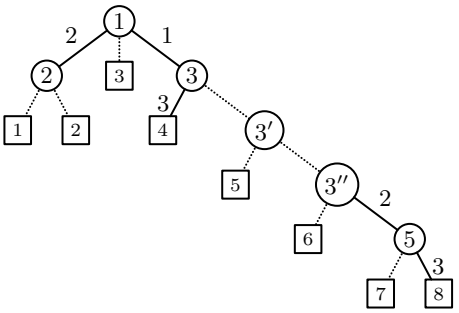
return $\bar{T}, \bar{d}, \bar{\chi}$



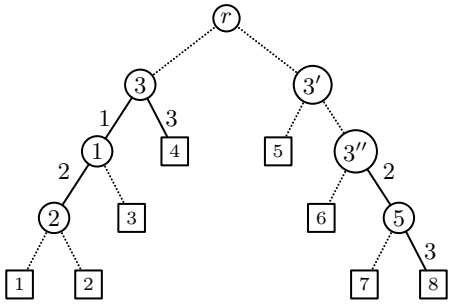
(a) Instance after inserting leaves (after step 4).



(b) Instance after contracting paths (after step 9).



(c) Instance after splitting vertices of degree greater than 3 (after step 17).



(d) Normalized instance.

Figure 4.3: Stages of the normalization done by algorithm NormalizeTree on the Darp instance shown in Figures 2.1(a) and 4.1. Dotted edges are edges of length 0. Other edge lengths are as indicated.

Proof. The key to the proof is the following observation:

> Every nontrivial component of $(V, A(L_m) \cup B')$ corresponds to one or more nontrivial components of $(\bar{V}, A(\bar{L}_m) \cup \bar{B}')$, which are at distance 0 from each other. (4.3)

This property can be verified by looking at how the changes done by NORMALIZE-TREE affect the balancing operation. Let us suppose that the "slots" are connected somehow by m requests.

The creation of leaves as source and destination vertices for the requests (steps 2ff) does not lead to further components, since balancing adds exactly one arc along edge $\{v, l_i\}$.

Similarly, the contraction of path segments (steps 5ff) does not increase the number of components either: All arcs traversing edges $\{u, v\}$ and $\{v, w\}$ have to traverse both successively, which is equivalent to one traversal of the new edge $\{u, w\}$. A similar argument holds for the addition of the root vertex r.

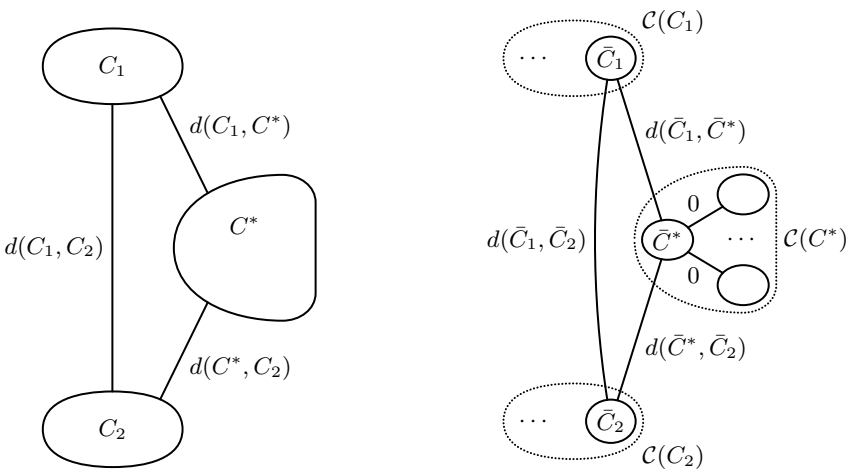
Finally, splitting vertices with degree larger than 3 (steps 10ff) may result in more nontrivial components: The new edge $\{v, v'\}$ need not be traversed by any request arc. In this case there is a new nontrivial component containing v', which is at distance 0 to the component of v.

Now we can use Property (4.3) to show that H' is a star metric whenever $\bar{H}'$ is. For the sake of simplicity, we will sometimes treat components of $(V, A(L_m) \cup B')$ as vertices of H' and similarly for $\bar{H}'$.

Let $\bar{C}^*$ be the center of the star metric of $\bar{H}'$ and C^* be the corresponding vertex in H'. Furthermore, suppose that C_1, C_2 are vertices of H' such that the distance $d(C_1, C_2)$ is determined by a shortest path starting in $u \in C_1$ and ending in $v \in C_2$. Since the length of shortest paths is not changed by NORMALIZETREE we have $d(C_1, C_2) = d(\bar{C}_1, \bar{C}_2)$ for some components $\bar{C}_1$ and $\bar{C}_2$ containing the vertices corresponding to u and v, respectively.

However, the star metric property of $\bar{H}'$ yields $d(\bar{C}_1, \bar{C}_2) = d(\bar{C}_1, \bar{C}^*) + d(\bar{C}^*, \bar{C}_2)$. By Property (4.3) we have $d(\bar{C}_1, \bar{C}^*) = d(C_1, C^*)$ and $d(\bar{C}^*, \bar{C}_2) = d(C^*, C_2)$ since lengths of shortest paths are preserved and the possibly remaining vertices of $\bar{H}'$ corresponding to C^* are all at distance 0 (see Figure 4.4). Thus $d(C_1, C_2) = d(C_1, C^*) + d(C^*, C_2)$ as claimed. □

We also claimed that the modified list random model is equivalent to the list random model conditioned on a fixed request-vertex-incidence vector χ.



(a) Situation in H'. (b) Situation in $\bar{H}'$.

Figure 4.4: Idea for translating star metric property from $\bar{H}'$ to H'. The symbol $\mathcal{C}(C)$ denotes the set of components of $(\bar{V}, A(\bar{L}_m) \cup \bar{B}')$ corresponding to a component C of $(V, A(L_m) \cup B')$.

Proposition 4.7 *Let χ be an arbitrary fixed request-vertex-incidence vector. The list random model and the modified list random model are related by*

$$\text{Prob}\left[A(x, \sigma) = \bar{A}\right] = \text{Prob}\left[A(L_m) = A \mid \chi(L_m) = \chi\right]. \tag{4.4}$$

Proof. The proof is along the lines of [13]. Let $A = \{a_1, \ldots, a_m\}$ be a request set with request-vertex-incidence vector χ. Furthermore, let k_i be the the multiplicity of a_i in A.

We start by computing $\text{Prob}\left[A(L_m) = A \mid \chi(L_m) = \chi\right]$. How many request lists L_m satisfy $A(L_m) = A$? The only difference between A and L_m is that A is unordered. For each request a_i a subset of k_i out of initially m positions in L_m can be chosen (without replacement), where all $k_i!$ subsets give rise to the same list L_m. Therefore there are in total

$$\binom{m}{k_1, \ldots, k_m} = \frac{m!}{\prod_{a_i \in A} k_i!}$$

different L_m with $A(L_m) = A$. Since L_m can equivalently be viewed as being an element of V^{2m} and for each $v \in V$ we have to choose χ_v out of initially $2m$ positions to obtain a L_m with request-vertex-incidence vector χ, the total number of such L_m is

$$\binom{2m}{\chi_{v_1}, \ldots, \chi_{v_n}} = \frac{(2m)!}{\prod_{v \in V} \chi_v!}.$$

We thus have

$$\text{Prob}\left[A(L_m) = A \mid \chi(L_m) = \chi\right] = \frac{m! \prod_{i=1}^{m} \chi_{v_i}!}{(2m)! \prod_{i=1}^{m} k_i!}.$$

Similarly, we have to determine the number of pairs (x, σ) leading to $\bar{A}$. Let $M(v) := \{l_1, \ldots, l_{\chi_v}\}$ be the set of leaves corresponding to v's "slots" and $A(v) := \{a_1, \ldots, a_{m'}\}$ the submultiset of request arcs incident to v. In order to generate $\bar{A}$, the permutation σ has to map $M(v)$ to $A(v)$ which can be done in

$$\binom{\chi_v}{k_1, \ldots, k_{m'}} = \frac{\chi_v!}{\prod_{a_i \in A(v)} k_i!}$$

distinct ways, since $|M(v)| = |A(v)| = \chi_v$. Considering all v, there are

$$\frac{\chi_{v_1}! \cdots \chi_{v_n}!}{\prod_{a_i \in A(v_1)} k_i! \cdots \prod_{a_i \in A(v_n)} k_i!} = \frac{\chi_{v_1}! \cdots \chi_{v_n}!}{\prod_{a_i \in A} (k_i!)^2}$$

different σs mapping each $M(v)$ to $A(v)$. Notice that σ implicitly determines whether a leaf is a "+"-leaf or a "−"-leaf, depending on whether it is mapped to a leaving or an entering request arc, respectively. In fact, the number of "+"s and "−"s in v's "slots" is already given by $\bar{A}$. However, if a_i is an arc of multiplicity k_i we still have $k_i!$ possibilities to map "+"-leaves to "−"-leaves.

Obviously, there are $\binom{2m}{m} m! = \frac{(2m)!}{m!}$ possibilities for (x, σ), so we get

$$\text{Prob}\left[A(x, \sigma) = \bar{A}\right] = \frac{m! \prod_{i=1}^{m} \chi_{v_i}!}{(2m)! \prod_{i=1}^{m} k_i!}.$$ □

The following lemma bounds the probability that a vertex is balanced.

Lemma 4.8 *For all $v \neq r$ we have*

$$\text{Prob}\left[S_v = 0\right] \leq 2\sqrt{\frac{6}{\pi}} \cdot \frac{1}{\sqrt{l(v)}} \in \mathcal{O}\left(l(v)^{-1/2}\right).$$

Proof. Notice that for $S_v = 0$ to hold, $l(v) =: k$ has to be even. Then

$$\begin{aligned}
\text{Prob}\left[S_v = 0\right] &= \frac{\binom{k}{k/2}\binom{2m-k}{m-k/2}}{\binom{2m}{m}} \\
&\overset{(A.3)}{\leq} \frac{\frac{2^k}{\sqrt{\pi k/2}} \frac{2^{2m-k}}{\sqrt{\pi(m-k/2)}}}{\frac{2^{2m-k}}{\sqrt{\pi m}}} \\
&= \frac{2\sqrt{\pi m}}{\pi\sqrt{k/2}\sqrt{m-k/2}}
\end{aligned}$$

Due to the choice of r (see Lemma 4.5) we know that both the left and the right subtree of r contain at least $\frac{2}{3}m$ leaves. Thus we have $k \leq \frac{2}{3} \cdot 2m$, leading to $m - k/2 \geq \frac{1}{3}m$ and

$$\text{Prob}\left[S_v = 0\right] \leq \frac{2\sqrt{m}\sqrt{3}}{\sqrt{\pi}\sqrt{k/2}\sqrt{m}} = 2\sqrt{\frac{6}{\pi}} \cdot \frac{1}{\sqrt{k}} \in \mathcal{O}\left(l(v)^{-1/2}\right). \qquad \square$$

4.4 Estimating the Number of Components Arising From Balancing Arcs

In this section it will be shown that $T_{B'}$ has less than one component per request a. a. s. To estimate the expected number of components we need to know a bound on the weight of all full binary trees for a certain weight function. The weights of each vertex are an upper bound for the probability that this vertex is the maximal vertex of component. This will become clearer in the proof of Lemma 4.10.

The next lemma, which is an improvement of Lemma 20 in [12] gives us this bound. We were able to improve this intermediate result in two ways: Most importantly, the proof we give here is much simpler and more intuitive than the original one. Moreover the constant α is a bit better than the original $\frac{49}{50} = 0.98$. Of course this is irrelevant for asymptotic results.

Lemma 4.9 *Let T be a full binary tree with l leaves. Then the vertex weight function $f\colon V \to \mathbb{R}$ defined by $f(v) = g(l(v))$ with*

$$g(k) := \begin{cases} \frac{1}{2} & k = 2 \\ \sqrt{\frac{2}{\pi k}} & k \geq 4,\ k \text{ even} \\ 0 & \text{otherwise} \end{cases}$$

satisfies $f(T) = \sum_{v \in V} f(v) \leq \alpha l$ for $\alpha \leq 0.42$.

Proof. We define

$$F(l) := \max\{f(T) \mid T \text{ is full binary tree with } l \text{ leaves}\}$$

to be the maximum weight possible for a full binary tree with l vertices and call

$$\varrho(l) := \frac{F(l)}{l}$$

the average weight of a maximum weight tree. It thus suffices to show that $\varrho(l) \leq \alpha \leq 0.42 \; \forall l$.

We assume that there are suitable constants $\alpha < 1$, $\beta > 0$ such that

$$\varrho(l) \leq (1 - \tfrac{\beta}{l})\alpha \tag{4.5}$$

holds for all l and derive conditions on α and β by an inductive argument in order to determine a small upper bound on α. The constant β is needed to obtain a bound for α which is less than $\frac{1}{2}$.

As a a basis for the induction, suppose we know values for α and β which satisfy (4.5) for all $l \leq l_0$. To see what happens for $l > l_0$, let T be a full binary tree with l vertices and root r. Denote by T_1 and T_2 the left and the right subtree having l_1 and l_2 leaves, respectively. We can thus write $f(T)$ as

$$f(T) = f(T_1) + f(T_2) + f(r) = f(T_1) + f(T_2) + g(l).$$

Since every tree T can be decomposed in this way we have

$$\begin{aligned} F(l) &= \max\{f(T) \mid T \text{ is a full binary tree with } l \text{ leaves}\} \\ &= \max\{f(T_1) + f(T_2) + g(l) \mid T = (T_1, r, T_2)\} \\ &= \max\{F(l_1) + F(l_2) \mid l_1 + l_2 = l\} + g(l). \end{aligned}$$

Using $\varrho(l_1)$ and $\varrho(l_2)$ and dividing by l we get

$$\varrho(l) = \frac{F(l)}{l} = \max\left\{ \frac{l_1\varrho(l_1) + l_2\varrho(l_2)}{l} \;\middle|\; l_1 + l_2 = l \right\} + \frac{g(l)}{l}.$$

We assumed that condition (4.5) is satisfied for any $l' \leq l_0$, so it is in particular satisfied for $\varrho(l_1)$ and $\varrho(l_2)$. Substitution yields

$$\begin{aligned} \varrho(l) &\leq \max\left\{ \frac{l_1(1 - \frac{\beta}{l_1})\alpha + l_2(1 - \frac{\beta}{l_2})\alpha}{l} \;\middle|\; l_1 + l_2 = l \right\} + \frac{g(l)}{l} \\ &= \max\left\{ \frac{\alpha\big((l_1 - \beta) + (l_2 - \beta)\big)}{l} \;\middle|\; l_1 + l_2 = l \right\} + \frac{g(l)}{l} \\ &= \max\left\{ \frac{\alpha(l_1 + l_2) - 2\alpha\beta}{l} \;\middle|\; l_1 + l_2 = l \right\} + \frac{g(l)}{l} \\ &= \alpha - \frac{2\alpha\beta - g(l)}{l} \\ &= \left(1 - \frac{2\beta - \frac{g(l)}{\alpha}}{l}\right)\alpha. \end{aligned}$$

We want α and β to satisfy (4.5), so we have the condition

$$2\beta - \tfrac{g(l)}{\alpha} \geq \beta$$

which is equivalent to $\beta \geq \frac{g(l)}{\alpha}$. Note that due to the definition of g, the value $g(l_0+2)$ is an upper bound for $g(l)$, since $l > l_0$.

All in all, α and β must obey the conditions

$$\begin{aligned} \varrho(l) &\leq (1 - \tfrac{\beta}{l})\alpha, \qquad \forall l \leq l_0 \\ \beta &\geq \frac{g(l_0+2)}{\alpha}. \end{aligned} \tag{4.6}$$

Setting

$$\beta := \beta(l_0) = \frac{g(l_0+2)}{\alpha}$$

Equation (4.6) reads

$$\varrho(l) \leq \alpha - \frac{g(l_0+2)}{l_0}.$$

If we set

$$\alpha(l_0) := \varrho(l_0) + \frac{g(l_0+2)}{l_0}$$

and $\alpha(l_0)$ and $\beta(l_0)$ satisfy (4.6) for all $l \leq l_0$, $\alpha(l_0)$ is an upper bound for α.

It turns out (and can be verified by a tedious computation) that $\alpha(6) \leq 0.42$ satisfies all conditions. This value could be improved but is already quite tight, as Table 4.1 shows. □

We are now ready to estimate the number of components in $T_{B'}$, which is done by using the random variables

$$X_j(x) := \text{number of components of } T_{B'} \text{ with maximal vertex in } V_j.$$

Clearly, the total number of components is $\sum_{j\geq 1} X_j(x)$.

We use a two step approach: We investigate $\sum_{j=1}^{j_0} X_j(x)$ and $\sum_{j>j_0} X_j(x)$ for $j_0 := \lfloor \log_2(2m)/3 \rfloor$ separately. This choice for j_0 is justified at the end of the proof for Lemma 4.10.

Lemma 4.10 *We have a. a. s.*

$$\sum\nolimits_{j=1}^{j_0} X_j(x) \leq (1 + o(1))2\alpha m.$$

l	$\varrho(l)$	$\alpha(l)$
2	0.25	0.449471
4	0.349736	0.431169
8	0.384997	0.416537
16	0.397464	0.409218
32	0.401872	0.406148
64	0.403430	0.404965
128	0.403981	0.404528
256	0.404176	0.404370
512	0.404245	0.404314
1024	0.404269	0.404294

Table 4.1: Some values for $\varrho(l)$ and $\alpha(l)$.

Proof. The technique to proof this result is the standard one: First we estimate the expectation $\mathbb{E}\left[\sum_{j=1}^{j_0} X_j(x)\right]$ and then we invoke a tail inequality — Azuma's inequality in this case — to obtain this concentration result. However, there is one technical obstruction to applying Azuma's inequality (see Theorem 3.11) directly: The components x_i of the vector x are not independent.

To overcome this difficulty, we make the following observation: The definition of the random variables and random objects $S_v(x)$, $E_{B'}(x)$, $T_{B'}(x)$, and $X_j(x)$ introduced so far do not depend on the fact that $\sum_i x_i = 0$. Consequently, we can extend their definition to an *arbitrary random* ± 1*-vector* $y \in \{-1,1\}^{2m}$ giving us the desired independence.

Let us consider a vector $y \in \{-1,1\}^{2m}$ chosen uniformly at random. The first step in the proof will be to show that $h(y) := \sum_{j=1}^{j_0} X_j(y)$ is $(1+o(1))2\alpha m$ a. a. s. In a second step this result is transfered to a restricted vector satisfying $\sum_i x_i = 0$. Furthermore let $Z_v(y)$ be the indicator variable

$$Z_v(y) := \begin{cases} 1 & v \text{ is the maximal vertex of a nontrivial component of } T_{B'}(y) \\ 0 & \text{otherwise} \end{cases}$$

for every $v \in V$.

$$\begin{aligned} \mathbb{E}\left[h(y)\right] \leq \mathbb{E}\left[\sum\nolimits_{j\geq 1} X_j(y)\right] &= \mathbb{E}\left[\sum\nolimits_{v\in V} Z_v(y)\right] \\ &\leq \sum\nolimits_{v\in V} \operatorname{Prob}\left[S_v(y)=0\right]. \end{aligned}$$

It is easy to see that

$$\begin{aligned} l(v) = 2 &\implies \text{Prob}\left[S_v(y) = 0\right] = \tfrac{1}{2} \\ l(v) > 2, l(v) \text{ even} &\implies \text{Prob}\left[S_v(y) = 0\right] = \binom{l(v)}{l(v)/2} 2^{-l(v)}, \end{aligned}$$

which using Lemma 4.9 gives the estimate

$$\begin{aligned} \sum\nolimits_{v \in V} \text{Prob}\left[S_v(y) = 0\right] &= \frac{|\{v \mid l(v) = 2\}|}{2} + \sum\nolimits_{v:\, l(v) \geq 4, v \text{ even}} \binom{l(v)}{l(v)/2} 2^{-l(v)} \\ &\overset{(A.3)}{\leq} \frac{|\{v \mid l(v) = 2\}|}{2} + \sqrt{\frac{2}{\pi}} \sum\nolimits_{v:\, l(v) \geq 4, v \text{ even}} l(v)^{-1/2} \\ &\leq f(T) \leq 2\alpha m. \end{aligned}$$

To show that $h(y)$ is concentrated around its expected value, let us consider two sequences $y, y' \in \{-1, 1\}^{2m}$ which differ only in position k.

We claim that the function h satisfies the Lipschitz-type condition

$$|h(y) - h(y')| \leq 2(2m)^{1/3}.$$

To see this, observe first that vertices $w \in V_{\leq j_0}$ which do not majorize the leaf l_k contribute in the same way to $h(y)$ and $h(y')$, since then $S_w(y) = S_w(y')$. Thus the only interesting vertices are those from the set

$$W := \{w \in V_{\leq j_0} \mid l_k \preceq w\}.$$

Note that W is a chain with respect to $\preceq$, because all $w \in W$ are on the path from l_k to the root r. The maximum vertex $\hat{w} := \max W$ majorizes at most 2^{j_0+1} leaves, so the subtree with root $\hat{w}$ has at most 2^{j_0+1} inner vertices. The difference $|h(y) - h(y')|$ can be bounded by the number of inner vertices, since at most each of them can be the maximum vertex of a nontrivial component. This leads to the claimed bound

$$|h(y) - h(y')| \leq 2^{j_0+1} \leq 2^{\log_2(2m)/3+1} = 2(2m)^{1/3}.$$

All components y_i of the vector y are independent random variables, so we can invoke Azuma's inequality (see Theorem 3.11) to get the bound

$$\text{Prob}\left[|h(y) - \mathbb{E}\left[h(y)\right]| \geq t\right] \leq 2\exp\left(-\frac{t^2}{2\sum_{i=1}^{2m}\left(2(2m)^{1/3}\right)^2}\right) = 2\exp\left(-\frac{t^2}{8(2m)^{5/3}}\right)$$

for arbitrary positive t.

Let us get back to constrained ± 1-sequences:

$$\begin{aligned}
\text{Prob}\left[\sum_{j=1}^{j_0} X_j \geq 2\alpha m + t\right] &= \text{Prob}\left[\left|h(y) - \mathbb{E}\left[h(y)\right]\right| \geq t \;\middle|\; \sum_{i=1}^{2m} y_i = 0\right] \\
&\leq \frac{\text{Prob}\left[\left|h(y) - \mathbb{E}\left[h(y)\right]\right| \geq t\right]}{\text{Prob}\left[\sum_{i=1}^{2m} y_i = 0\right]} \\
&\leq 2\exp\left(-\frac{t^2}{8(2m)^{5/3}}\right)\frac{2^{2m}}{\binom{2m}{m}} \\
&\sim 2\sqrt{\pi m}\exp\left(-\frac{t^2}{8(2m)^{5/3}}\right).
\end{aligned}$$

In order to prove the lemma, it remains to choose t such that

1. $t \in o(m)$ (left hand side)
2. $2\exp\left(-\frac{t^2}{8(2m)^{5/3}}\right)\sqrt{\pi m} \in o(1)$ (right hand side).

Set $f(m) := 1/(8(2m)^{5/3})$. Clearly, the second condition can be satisfied if

$$\begin{aligned}
& 2\exp\left(-f(m)\right)^{t^2}\sqrt{\pi m} \ll 1 \\
\iff \quad & t^2 \ln(\exp\left(-f(m)\right)) \ll \ln(\sqrt{\pi m})^{-1} \\
\iff \quad & -t^2 f(m) \ll -\ln(\sqrt{\pi m}) \\
\iff \quad & t^2 f(m) \gg \ln(\sqrt{\pi m}) \\
\iff \quad & t^2 \gg \frac{\ln(\sqrt{\pi m})}{f(m)}.
\end{aligned}$$

We choose $t^2 = \frac{\ln(2m)}{f(m)} = 8\ln(2m)(2m)^{5/3}$, satisfying both conditions.

Note that for $j_0 := \lfloor c\log_2(2m)\rfloor$ our bound for $|h(y) - h(y')|$ would be $2(2m)^c$, leading to

$$\text{Prob}\left[\left|h(y) - \mathbb{E}\left[h(y)\right]\right| \geq t\right] \leq 2\exp\left(-\frac{t^2}{8(2m)^{5c}}\right).$$

Because of the second condition, t^2 has to be greater than $(2m)^{5c}$, implying $t \geq (2m)^{5c/2}$. Our choice $c = \frac{1}{3}$ is just small enough to meet the first condition for t, i. e., that $t \in o(m)$. □

The following lemma estimates the number of components of $T_{B'}$ with maximal vertex in $V_{>j_0}$. Combining it with the preceding lemma we see that the total number of components is a. a. s.

$$\sum\nolimits_j X_j(x) \leq 2\alpha m.$$

Lemma 4.11 *We have a. a. s.*

$$\sum\nolimits_{j_0<j} X_j(x) \in o(m).$$

Proof. Similarly to the argument in Lemma 4.10 we have the estimate

$$\begin{aligned}\mathbb{E}\left[\sum\nolimits_{j_0<j} X_j(x)\right] &= \mathbb{E}\left[\sum\nolimits_{v\in V_{>j_0}} Z_v(x)\right]\\ &\le \sum\nolimits_{v\in V_{>j_0}} \mathrm{Prob}\left[S_v(x)=0\right]\\ &\le 2\sqrt{\frac{6}{\pi}}\sum\nolimits_{v\in V_{>j_0}} \frac{1}{\sqrt{l(v)}},\end{aligned}$$

where the last inequality has been obtained by using Lemma 4.8.

Next, we rewrite the last term to sum by the number $l(v)$ of leaves below a vertex v instead of vertices v itself. Note that each vertex $v \in V_{>j_0}$ majorizes at least $2^{j_0+1} \ge 2^{\log_2(2m)/3} = (2m)^{1/3}$ and at most $2m$ leaves.

$$2\sqrt{\frac{6}{\pi}}\sum\nolimits_{v\in V_{>j_0}} \frac{1}{\sqrt{l(v)}} = 2\sqrt{\frac{6}{\pi}}\sum\nolimits_{b=(2m)^{1/3}}^{2m} \frac{1}{\sqrt{b}}\cdot\left|\{v\in V_{>j_0} \mid l(v)=b\}\right|$$

Since all vertices v with $l(v) = b$ for a given b form an antichain with respect to $\preceq$, there are at most $\frac{2m}{b}$ of them:

$$\begin{aligned}&\le 2\sqrt{\frac{6}{\pi}}\sum\nolimits_{b=(2m)^{1/3}}^{2m} \frac{1}{\sqrt{b}}\cdot\frac{2m}{b}\\ &= 4m\sqrt{\frac{6}{\pi}}\sum\nolimits_{b=(4m)^{1/3}}^{2m} b^{-3/2}\\ &\le 4m\sqrt{\frac{6}{\pi}}\int_{b=(2m)^{1/3}-1}^{2m} b^{-3/2}\,\mathrm{d}b\\ &\le 4m\sqrt{\frac{6}{\pi}}\cdot 2(2m)^{-1/6}\\ &= 8\sqrt{\frac{6}{\pi}}\frac{m}{(2m)^{1/6}} \in o(m).\end{aligned}$$

Invoking Markov's inequality with $t := cm^{5/6}\log m \in o(m)$ establishes the "high probability" claim:

$$\begin{aligned}\mathrm{Prob}\left[\sum\nolimits_{j_0<j} X_j(x) \ge cm^{5/6}\log m\right] &\le \frac{\mathbb{E}\left[\sum_{j_0<j} X_j(x)\right]}{cm^{5/6}\log m}\\ &\le \frac{1}{\log m} \in o(1).\end{aligned}$$

□

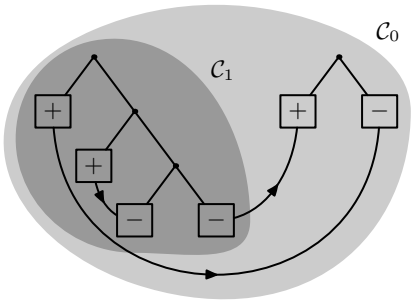


Figure 4.5: Illustration of separated sets: The set $\mathcal{C}_0$ is separated in $T_{AB'}$, since there are no arcs in A which connect its components to components not in $\mathcal{C}_0$. In contrast, the set $\mathcal{C}_1$ is clearly not separated in $T_{AB'}$.

4.5 The Structure of the Graph After Balancing

We already know that the graph $T_{B'}$ a. a. s. has no more than $2\alpha m$ components, the set of which will be denoted by $\mathcal{C}_{B'}$. We will now consider the graph $T_{AB'} := (V, E_{B'} \cup E_A)$ in which the components of $T_{B'}$ are joined by request edges.

A subset $\mathcal{C}_0 \subseteq \mathcal{C}_{B'}$ will be called *separated (in $T_{AB'}$)*, if there is no edge in E_A joining a vertex in $\mathcal{C}_0$ with a vertex outside $\mathcal{C}_0$ (see Figure 4.5).

It is an easy fact that each component of $T_{AB'}$ is a nonempty separated subset of $\mathcal{C}_{B'}$, which is minimal w. r. t. inclusion. To see this, let $\mathcal{C}_0$ be such a subset. Since $\mathcal{C}_0$ is minimal w. r. t. inclusion, for each component $C \in \mathcal{C}_0$, the subset $\mathcal{C}_0 \setminus \{C\}$ is not separated, which tells us that the edges in E_A join *exactly* all components in $\mathcal{C}_0$. Thus the subgraph of $T_{AB'}$ induced by $\mathcal{C}_0$ is obviously a component of $T_{AB'}$. Now let C be a component of $T_{AB'}$ and denote by $\mathcal{C}_{B'}(C)$ the set of components of $T_{B'}$ it is made of. Clearly, $\mathcal{C}_{B'}(C)$ is nonempty and separated. Furthermore, it is also minimal w. r. t. inclusion, because if $\mathcal{C}_{B'}(C) \setminus \{C'\}$ was separated for some $C' \in \mathcal{C}_{B'}(C)$, C cannot be a component.

We will use this equivalence between the components of $T_{AB'}$ and the minimal separated subsets of $\mathcal{C}_{B'}$ by bounding the number of separated subsets and thus the number of components. Denote by $l^+(\mathcal{C}_0)$ the number of "+"-leaves in $\mathcal{C}_0$, i. e., the number of leaves l_i with $x_i = +1$.

Lemma 4.12 *Suppose x_0 is such that $|\mathcal{C}_{B'}(x_0)| \leq 2\alpha m$. The expected number of separated subsets $\mathcal{C}_0 \subseteq \mathcal{C}_{B'}(x_0)$ satisfying*

- $|\mathcal{C}_0| = k$ *and*
- $l^+(\mathcal{C}_0) \leq \frac{m}{2}$

is at most

$$(2\alpha)^k \frac{(k+l)_l}{(m-k)_l}. \tag{4.7}$$

where we assumed that $l^+(\mathcal{C}_0) = k + l$. *Note that* $l \geq 0$ *since every* $C \in \mathcal{C}_{B'}(x_0)$ *has at least one "+"-leaf.*

Moreover, Equation (4.7) bounds the expected number of such $\mathcal{C}_0$ *even for arbitrary* l, $k + l \leq l^+(\mathcal{C}_0)$.

Proof. Consider for each subset $\mathcal{C}_0$ with the above properties the indicator variable

$$I_{\mathcal{C}_0} := \begin{cases} 1 & \mathcal{C}_0 \text{ is separated} \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, $\mathcal{C}_0$ is separated if and only if all its "+"-leaves are connected to its "−"-leaves:

$$\begin{aligned} \text{Prob}\left[I_{\mathcal{C}_0} = 1\right] = \frac{l^+(\mathcal{C}_0)!(m - l^+(\mathcal{C}_0))!}{m!} &= \binom{m}{l^+(\mathcal{C}_0)}^{-1} \\ &\leq \binom{m}{k+l}^{-1} \quad \text{for } l \text{ with } k + l \leq l^+(\mathcal{C}_0). \end{aligned}$$

The last inequality follows from the fact that $l^+(\mathcal{C}_0) \leq \frac{m}{2}$.

We can now use the bound for the probability to estimate the expected number of separated subsets $\mathcal{C}_0$:

$$\begin{aligned} \mathbb{E}\left[\sum_{\mathcal{C}_0 \subseteq \mathcal{C}_{B'}, |\mathcal{C}_0| = k, l^+(\mathcal{C}_0) \leq \frac{m}{2}} I_{\mathcal{C}_0}\right] &\leq \sum_{|\mathcal{C}_0| = k, l^+(\mathcal{C}_0) \leq \frac{m}{2}} \binom{m}{k+l}^{-1} \\ &\leq \binom{|\mathcal{C}_{B'}|}{k} \binom{m}{k+l}^{-1} \\ &= \frac{(|\mathcal{C}_{B'}|)_k \, (k+l)!}{k! \, (m)_{k+l}} \\ &\leq \frac{(2\alpha m)_k \, (k+l)_l}{(m)_{k+l}} \\ &= \frac{(2\alpha)^k (m)(m - \frac{1}{2\alpha}) \cdots (m - \frac{k+1}{2\alpha}) \, (k+l)_l}{(m)_{k+l}} \\ &\leq \frac{(2\alpha)^k \, (m)_k \, (k+l)_l}{(m)_{k+l}} \qquad (\text{since } 2\alpha \leq 1) \\ &= (2\alpha)^k \frac{(k+l)_l}{(m-k)_l}. \end{aligned}$$

This completes the proof. □

Let us call a component of $T_{AB'}$ *large* if it contains more than m leaves and *small* otherwise. Note that this implies that there is at most one large component. We are now ready to prove the following result on the structure of $T_{AB'}$.

Proposition 4.13 *The graph $T_{AB'}$ enjoys the following properties a. a. s.:*

1. *The are $\mathcal{O}(\log m)$ small components.*
2. *All small components are cycles of length $\mathcal{O}(\log m)$.*
3. *There is one large component containing $(2 - o(1))m$ leaves.*

Proof. Assume that $|\mathcal{C}_{B'}| \leq 2\alpha m$. By definition, a subset $\mathcal{C}_0 \subseteq \mathcal{C}_{B'}$ is separated if and only if $\mathcal{C}_{B'} \setminus \mathcal{C}_0$ is separated. Therefore for each minimal separated subset $\mathcal{C}_0$with $l^+(\mathcal{C}_0) \leq \frac{m}{2}$ there is at most one other minimal separated subset $\mathcal{C}_0'$ with $l^+(\mathcal{C}_0') > \frac{m}{2}$. The number $\eta(T_{AB'})$ of components of $T_{AB'}$ is thus bounded by twice the number of minimal separated subsets $\mathcal{C}_0$ with $l^+(\mathcal{C}_0) \leq \frac{m}{2}$, which in turn is bounded by the number of separated subsets with $l^+(\mathcal{C}_0) \leq \frac{m}{2}$.

$$\begin{aligned}\mathbb{E}\left[\eta(T_{AB'})\right] &\leq 2\sum\nolimits_{k=1}^{2\alpha m} \mathbb{E}\left[\left|\{\mathcal{C}_0 \mid \mathcal{C}_0 \text{ is separated, } |\mathcal{C}_0| = k,\, l^+(\mathcal{C}_0) \leq \tfrac{m}{2}\}\right|\right] \\ &\leq 2\sum\nolimits_{k=1}^{\infty} (2\alpha)^k \in \mathcal{O}(1)\,.\end{aligned}$$

The second inequality is obtained by invoking Lemma 4.12 with $l = 0$. The Markov inequality yields for $t := c \log m$

$$\text{Prob}\left[\eta(T_{AB'}) \geq c \log m\right] \leq \frac{\mathcal{O}(1)}{c \log m} \in o(1) \qquad \text{as } m \to \infty,$$

thereby establishing claim 1.

Lemma 4.12 allows us to conclude that a. a. s. every small component of $T_{AB'}$ consists of $\mathcal{O}(\log m)$ nontrivial components of $T_{B'}$. Suppose $k \geq c \log m$ for some suitably large c, leading to

$$\begin{aligned}\mathbb{E}\left[\left|\{\mathcal{C}_0 \mid \mathcal{C}_0 \text{ is separated, } |\mathcal{C}_0| = k,\, l^+(\mathcal{C}_0) \leq \tfrac{m}{2}\}\right|\right] &\leq (2\alpha)^k \frac{(k+l)_l}{(m-k)_l} \\ &\leq (2\alpha)^{c\log m} \frac{(k+l)_l}{(m-k)_l} \\ &\leq m^{-\bar{c}} \frac{(m/2)_l}{(m/2)_l} \qquad \text{for some } \bar{c} > 0 \\ &= m^{-\bar{c}} \in o(1) \qquad \text{as } m \to \infty.\end{aligned}$$

We used the fact $k+l \leq \frac{m}{2}$ and the implied bound $m-k \geq \frac{m}{2}$ in this estimate. By the first moment method, there are a. a. s. no small components consisting of more than $c \log m$ nontrivial components.

So let us assume $k \leq c \log m$. Again, we can use Lemma 4.12 and the first moment method to show that $l^+(\mathcal{C}_0) = k$ a. a. s.. The expected number of separated sets $\mathcal{C}_0$ having $l^+(\mathcal{C}_0) - k =: l > 0$ is at most

$$(2\alpha)^k \frac{(k+l)_l}{(m-k)_l} \leq \frac{(k+l)_l}{(m-k)_l} \in \mathcal{O}\left(m^{-1}\right).$$

This can be seen by considering two cases:

Case 1: l is small: $l \leq c \log m$

We then have

$$\frac{(k+l)_l}{(m-k)_l} \leq \frac{k+l}{m-k} \sim \frac{1}{m}.$$

Case 2: l is large: $l \geq c \log m$

$$\begin{aligned}
\frac{(k+l)_l}{(m-k)_l} &\sim \frac{(k+l)_l}{(m)_l} \\
&\leq \frac{(m/2)_l}{(m)_l} \\
&= \prod_{i=0}^{l-1} \frac{m/2-i}{m-i} \\
&= \prod_{i=0}^{l-1} \left(\frac{1}{2} - \frac{i}{2(m-i)}\right) \\
&\leq \left(\frac{1}{2}\right)^l \leq \left(\frac{1}{2}\right)^{c \log m} = \frac{1}{m^{\bar{c}}} \qquad \text{for some } \bar{c} > 1.
\end{aligned}$$

The condition $l^+(\mathcal{C}_0) = k$ implies that all nontrivial components of $T_{B'}$ that make up $\mathcal{C}_0$ are "cherry"-like structures with one "+"-leaf and one "−"-leaf each, so $\mathcal{C}_0$ must be a cycle, which was claim 2.

We now know that there are at most $\mathcal{O}(\log m)$ small components, each having at most $\mathcal{O}(\log m)$ leaves, so the remaining $2m - \mathcal{O}\left(\log^2 m\right)$ leaves have to be in the unique large component. □

4.6 Algorithm USE-MST Is a. a. s. Optimal

Throughout this section we implicitly assume that the properties stated in Proposition 4.13 hold. We want to show that the component graph of $T_{AB'}$ will be a star

metric. We do this by showing that every path connecting different small components passes through the large component C^*. The main technical idea is to identify larger regions in $T_{AB'}$ which border single small components, so different small components will be in different regions. Furthermore crossing region borders will lead a. a. s. to a traversal of C^*.

The probability that a certain leaf set associated with such a region encompasses leaves from more than one small component is estimated in the following technical lemma.

Lemma 4.14 *Let $M \subseteq \{l_1, \ldots, l_{2m}\}$ be a subset of $l := |M|$ leaves. Consider distinct small components $C_1, C_2 \in \mathcal{C}_{B'}$ which both contain leaves of M. Then*

$$\text{Prob}\left[\exists C_1, C_2 \in \mathcal{C}_{B'} \textit{ which end up in different } \text{small } \textit{components}\right] \in \mathcal{O}\left(\tfrac{l^2}{m^2}\right).$$

Proof. Three cases are possible:

1. C_1, C_2 are joined to further distinct components of $T_{B'}$
2. C_1 makes up a nontrivial component of $T_{AB'}$ while C_2 is joined to further components of $T_{B'}$
3. C_1, C_2 are both nontrivial components of $T_{AB'}$

Case 1: C_1, C_2 are joined to further distinct components of $T_{B'}$

Let $\mathcal{C}_1, \mathcal{C}_2$ be the subsets of $T_{B'}$'s components C_1 and C_2 are joined to, respectively. Clearly, $\mathcal{C}_0 := \{C_1, C_2\} \cup \mathcal{C}_1 \cup \mathcal{C}_2$ is a separated subset with $k := |\mathcal{C}_0| \geq 4$ components. The probability for this case can thus be bounded by the probability that there is such a $\mathcal{C}_0$. For a fixed $\mathcal{C}_0$, the probability that $\mathcal{C}_0$ is separated is at most $\binom{m}{k}^{-1}$. Furthermore there are at most $\binom{l}{2}$ choices for $\{C_1, C_2\}$ and at most $\binom{|\mathcal{C}_{B'}|}{k-2}$ possibilities for $\mathcal{C}_1$ and $\mathcal{C}_2$.

$$\begin{aligned}
&\text{Prob}\left[\exists C_1, C_2 \text{ ending up in different } \textit{small} \text{ components}\right] \\
&\leq \sum_{k=4}^{\mathcal{O}(\log m)} \binom{l}{2}\binom{|\mathcal{C}_{B'}|}{k-2}\binom{m}{k}^{-1} \\
&\leq \sum_{k=4}^{\mathcal{O}(\log m)} \frac{l^2}{2}\frac{(2\alpha m)_{k-2}}{(k-2)!}\frac{k!}{(m)_k} \\
&\leq \sum_{k=4}^{\mathcal{O}(\log m)} \frac{l^2 k^2}{(m-k+2)(m-k+1)}\frac{(2\alpha m)_{k-2}}{(m)_{k-2}} \\
&\leq \sum_{k=4}^{\mathcal{O}(\log m)} \frac{l^2 k^2}{(m-k+1)^2}(2\alpha)^k \\
&\leq \frac{l^2}{(m-\mathcal{O}(\log m))^2}\sum_{k=4}^{\mathcal{O}(\log m)} k^2(2\alpha)^k \in \mathcal{O}\left(\tfrac{l^2}{m^2}\right).
\end{aligned}$$

Case 2: C_1 is a nontrivial component of $T_{AB'}$, C_2 is joined to further components of $T_{B'}$

Similarly to the previous case, let $\mathcal{C}_2$ be the subset of $T_{B'}$'s components C_2 is joined to. We set $\mathcal{C}_0 := \{C_2\} \cup \mathcal{C}_2$ and $k := |\mathcal{C}_0| \geq 2$. This time, there are at most $l(l-1)$ choices for $\{C_1, C_2\}$ and not more than $\binom{|\mathcal{C}_{B'}|}{k-1}$ choices for $\mathcal{C}_2$. The probability that both $\{C_1\}$ and $\mathcal{C}_0$ are separated is

$$\frac{l^+(C_1)!\, l^+(\mathcal{C}_0)!\, (m - l^+(C_1) - l^+(\mathcal{C}_0))!}{m!},$$

leading to the estimate

$$\begin{aligned}
&\text{Prob}\left[\exists C_1, C_2 \text{ ending up in different } \textit{small} \text{ components}\right] \\
&\leq \sum\nolimits_{k=2}^{\mathcal{O}(\log m)} l(l-1)\binom{|\mathcal{C}_{B'}|}{k-1}\frac{l^+(C_1)!\, l^+(\mathcal{C}_0)!\, (m - l^+(C_1) - l^+(\mathcal{C}_0))!}{m!} \\
&\leq \sum\nolimits_{k=2}^{\mathcal{O}(\log m)} l^2\binom{|\mathcal{C}_{B'}|}{k-1}\frac{k!(m-1-k)!}{m!} \\
&\leq \sum\nolimits_{k=2}^{\mathcal{O}(\log m)} \frac{l^2}{m-k}\frac{(2\alpha m)_{k-1}}{(k-1)!}\frac{k!}{(m)_k} \\
&\leq \sum\nolimits_{k=2}^{\mathcal{O}(\log m)} \frac{l^2 k}{(m-k)(m-k+1)}\frac{(2\alpha m)_{k-1}}{(m)_{k-1}} \\
&\leq \frac{l^2}{(m - \mathcal{O}(\log m))^2}\sum\nolimits_{k=2}^{\mathcal{O}(\log m)} k(2\alpha)^{k-1} \in \mathcal{O}\left(\tfrac{l^2}{m^2}\right).
\end{aligned}$$

Case 3: C_1 and C_2 are both nontrivial components of $T_{AB'}$

In the final case we have the bound

$$\begin{aligned}
&\text{Prob}\left[\exists C_1, C_2 \text{ ending up in different } \textit{small} \text{ components}\right] \\
&\leq \binom{l}{2}\frac{l^+(C_1)!\, l^+(C_2)!\, (m - l^+(C_1) - l^+(C_2))!}{m!} \\
&\leq \frac{l^2}{2}\frac{(m-2)!}{m!} = \frac{l^2}{2m(m-1)} \in \mathcal{O}\left(\tfrac{l^2}{m^2}\right).
\end{aligned}$$

□

For the final proof of Theorem 4.2 we need the following proposition, which actually identifies the "regions" mentioned above.

Proposition 4.15 *The graph $T_{AB'}$ enjoys the following properties a. a. s.:*

1. *There is no vertex $v \in V_{\leq 0.9\log_2(2m)}$ that majorizes vertices of distinct small components.*

2. *If $u, v \in V_{\geq 0.9\log_2(2m)}$ are such that $u \preceq v$, u majorizes vertices of a small component C_u and v majorizes vertices of another small component $C_v \neq C_u$, then there is a vertex s, $u \preceq s \preceq v$, with $|S_s| > 2$.*

Proof. For the sake of brevity let $j_0 := 0.9\log_2(2m)$ and $l_0 := 2^{j_0} = (2m)^{0.9}$. The first claim is a relatively easy consequence of Lemma 4.14 applied to the leaf set $M := \{l_i \in V \mid l_i \preceq v\}$ for some $v \in V_{\leq j_0}$. It suffices to show that there are no *maximal* vertices in $V_{\leq j_0}$ majorizing distinct small components. Lemma 4.4 tells us that there are at most $\frac{2m}{2^j}$ maximal vertices in V_j. For a fixed j, $1 \leq j \leq j_0$, the expected number of maximal vertices majorizing distinct small components thus does not exceed (note that $l := |M| \leq 2^{j+1}$)

$$\frac{2m}{2^j}\mathcal{O}\left(\left(\frac{2^{j+1}}{m}\right)^2\right) = \mathcal{O}\left(\frac{2^{j+1}}{m}\right) \in \mathcal{O}\left(m^{-0.1}\right).$$

There are only $\log_2(2m)$ choices for j, so the expected number of $v \in V_{\leq j_0}$ is $\mathcal{O}\left(m^{-0.1}\log m\right) \in o(1)$, entailing claim 1.

Establishing the second claim is more subtle. We think of the vertex set $V_{\geq j_0}$ as being partitioned into *segments* which are paths from the minimal vertices towards the root. A *segment* is a path $v_i \cdots v_i'$ and will be denoted by $[v_i, \ldots, v_i']$. The basic idea is that for any segment $[v_i, \ldots, v_i']$ we have that

1. there is a. a. s. at most *one* small component majorized by v_i' and not by v_i, and

2. $|S_{v_i}| > 2$ a. a. s.

In that situation the path from u to v has to leave the segment $[v_1, \ldots, v_1']$ (which is the first segment on the path from u to v majorizing C_u) and to enter another segment $[v_2, \ldots, v_2']$ through vertex v_2 satisfying $|S_{v_2}| > 2$ – we have thus found our s.

To show that the mentioned properties hold a. a. s., we require that the partitioning is such that

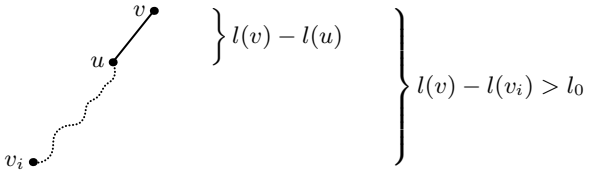
- Each segment $[v_i, \ldots, v_i']$ majorizes at most l_0 *new* small leaves, i. e., the number of leaves majorized by v_i' but not by v_i is at most l_0:

$$l(v_i') - l(v_i) \leq l_0. \tag{4.8}$$

- The number of leaves majorized by v_{i+1} but not by v_i is at least $\frac{l_0}{2}$, where v_i and v_{i+1} are initial vertices of successive segments:

$$l(v_{i+1}) - l(v_i) \geq \tfrac{l_0}{2}. \tag{4.9}$$

Algorithm PARTITIONUPPERPART (see Algorithm 4.2) shows how to achieve this. The critical issue is to ensure that the number of leaves majorized by initial vertices of successive segments grows at least by $\frac{l_0}{2}$, i. e., establishing condition (4.9) which is done in steps 9ff. The situation at this point is as follows:



There are the following two cases:

Case 1: $l(v) - l(u) > \frac{l_0}{2}$

$$l(v_{i+1}) - l(v_i) = l(v) - l(v_i) \geq l(v) - l(u) > \frac{l_0}{2}.$$

Case 2: $l(v) - l(u) \leq \frac{l_0}{2}$

We need that $l(v) - l(v_i) > l_0$ which is the case due to the choice of v controlled by the while-loop. We then have

$$\begin{aligned} l(v_{i+1}) - l(v_i) = l(u) - l(v_i) &= l(v) - l(v_i) - \big(l(v) - l(u)\big) \\ &> l_0 - \frac{l_0}{2} \geq \frac{l_0}{2}. \end{aligned}$$

Now that we are convinced that a partition of $V_{\geq j_0}$ into segments meeting the requirements (4.8) and (4.9) exists we turn to verifying that there is a. a. s. at most one small component majorized by v_i' but not by v_i in a fixed segment $[v_i, \ldots, v_i']$. By invoking Lemma 4.14 with $M := \{l_j \mid l_j \preceq v_i', l_j \not\preceq v_i\}$ we see that the probability that the leaves majorized by v_i' but not by v_i belong to more than one small component is at most $\mathcal{O}\left(\frac{l_0^2}{m^2}\right)$.

We still need to show that the probability for $|S_{v_i}| \leq 2)$ is small. We have the estimate

$$\text{Prob}\left[|S_{v_i}| = 0\right] + \text{Prob}\left[|S_{v_i}| = 2\right] \in \mathcal{O}\left(l(v_i)^{-1/2}\right) \subseteq \mathcal{O}\left(l_0^{-1/2}\right),$$

by means of Lemma 4.8 and the fact that $\text{Prob}\left[S_{v_i} = \pm 2\right] \sim \text{Prob}\left[S_{v_i} = 0\right]$.

It remains to estimate the probability that both properties hold for *all* segments. To this end we need to know the number of segments. condition (4.9) allows us to bound the total number of segments as follows.

- There are at most $|\{\text{minimal vertices of } V_{\geq j_0}\}|$ "upper" segments containing at most $\frac{l_0}{2}$ new leaves, i. e., at most $\frac{2m}{2^{j_0}} = \frac{2m}{l_0}$ ones (by Lemma 4.4).
- There are at most $\frac{2m}{l_0/2} = \frac{4m}{l_0}$ other segments.

All in all, there are not more than $\frac{6m}{l_0}$ segments.

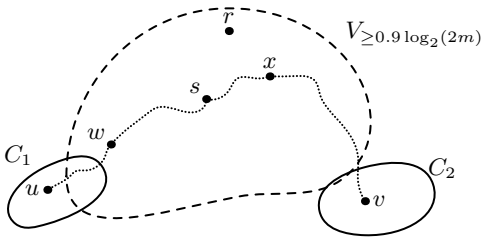
The expected number of segments violating our requirements is then at most

$$\frac{6m}{l_0}\left(\mathcal{O}\left(\frac{l_0^2}{m^2}\right) + \mathcal{O}\left(l_0^{-1/2}\right)\right) = \mathcal{O}\left(\frac{l_0}{m} + \frac{m}{l_0^{-3/2}}\right) = o(1) \quad \text{as } m \to \infty,$$

which completes the proof. □

Proof (of Theorem 4.2). The properties of $T_{AB'}$ stated in Propositions 4.13 and 4.15 hold a. a. s., so we may presume them. Our goal is to show that any path from a small component C_1 to a small component C_2 passes through the large component C^*, which implies that the component graph is a star metric. By Proposition 2.11 the proof will be complete.

Suppose u is a vertex of C_1 and v is a vertex of C_2. If $u \in V_{\leq 0.9\log_2(2m)}$ then the first part of Lemma 4.15 guarantees us that the path to v crosses $V_{\geq 0.9\log_2(2m)}$ since there is no other small component below the maximal vertex of $V_{\leq 0.9\log_2(2m)}$ majorizing u. Otherwise, u is in $V_{\geq 0.9\log_2(2m)}$ so the path connecting u and v passes through $V_{\geq 0.9\log_2(2m)}$ in any case. Let w be the first vertex on the path from u to v which is in $V_{\geq 0.9\log_2(2m)}$ and x be the maximal vertex of the path. We thus have more or less the following situation:



Most importantly, vertices w and x majorize different small components C_1 and C_2 and x majorizes w, so we can apply the second part of Proposition 4.15 to see that there is a vertex s satisfying $w \preceq s \preceq x$ and $|S_s| > 2$. Since the condition $|S_s| > 2$ can only be met by vertices outside a cycle, the vertex s has to be in C^*. □

Algorithm 4.2 Algorithm PARTITIONUPPERPART for partitioning the vertex set $V_{\geq 0.9\log_2(2m)}$ of a tree T into segments.

PARTITIONUPPERPART(T)

Input: A full binary Tree $T = (V, E)$.

Output: Set of segments S.

```
1  Mark the root r and add [r] to S.
2  for v̌ ∈ V_{≥j0}, v̌ minimal do
3    i := 0; v_i := v̌; v := ϱ(v̌);
4    repeat
5      while v is not marked and l(v) − l(v_i) ≤ l_0 do
6        v := ϱ(v);
7      end while
8      Let u be the predecessor of v on the path v_i ··· v.
       { Ensure that between v_i and v_{i+1} there are at least l_0/2 new leaves: }
9      if l(v) − l(u) > l_0/2 then
10       v_{i+1} := v;
11     else
12       v_{i+1} := u;
13     end if
14     Let v'_i be the predecessor of v_{i+1} on the path v_i ··· v_{i+1}.
15     Add new segment [v_i, ..., v'_i] to S and mark all of its vertices.
16     i := i + 1;
17   until v_i is marked
18 end for
   return S
```

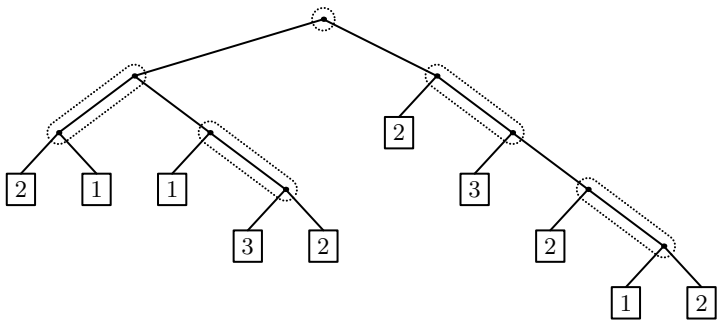


Figure 4.6: The set $V_{\geq j_0}$ of a tree and the segmentation output by PARTITIONUPPERPART if minimal vertices are processed from left to right (we assume values $j_0 = 2$ and $l_0 = 6$). The leaves are maximal vertices of $V_{<j_0}$ giving the number of "real" leaves they majorize.

5 Towards Probabilistic Competitive Analysis of the OnlineDarp on Trees

Until now we totally neglected the *online nature* of typical Dial-a-Ride problems. We start to overcome this deficiency in this chapter.

First we introduce a natural extension of the offline Darp to get an online version. We then review the standard tool for analyzing online algorithms, namely *competitive analysis*, in Sections 5.2 and 5.3 based on the presentation in the book of Borodin and El Yaniv [9]. This notion is then extended to yield *probabilistic competitive analysis*. We also give some lower bounds for the performance of any algorithm for the OnlineDarp first published by Ascheuer et al. [2].

The same source mentions two *online strategies* called Replan and Ignore, which allow to build online algorithms from *any* algorithm for the offline problem. Both strategies are discussed and used to obtain online algorithms based on use-MST. We will describe how they influence the stochastic structure of the offline problems they have to solve.

Finally, a special case of the OnlineDarp on trees is considered and analysed in Section 5.6 under the assumption that the requests arrive faster than they can be served. This unusual assumption will be discussed in that section, too.

The results on the structure of offline problems solved by Ignore and Replan, the notion of probabilistic competitiveness as well as the first such competitiveness result in Section 5.6 are new contributions created in cooperation with Sven O. Krumke.

5.1 Online Versions of Darp

So far we only considered the *offline* Darp, i. e., all requests were known in advance and we could in principle predict the entire future of any solution and thus determine an optimal one. However, in practice the requests arrive distributed over longer time

periods. For reasons discussed already in the introduction the requests must not be delayed too long and we have to come up with intermediate schedules for some subset of the requests. In other words, we try to solve an *online problem*.

The further difficulty introduced by this online aspect is the lack of information about the future. Typically, after the arrival of some new requests the so-far optimal schedule may be very bad w. r. t. the new extended request set. Clearly, we could avoid this effect if we could anticipate all future requests.

We will not give a formal online computation model for our notions of online and offline algorithms, but use more intuitive definitions. However, a formalization of the online version of Darp in the framework of *request-answer-games* can be found, for example, in [28]. All notions can be suitably generalized to fit general online problems.

Before turning to performance measures for such online problems in the next sections, we first give a online version of Darp and try to model the evolution of the request set. To this end, we equip each request r_i with a *release time* t_i which is the time the request enters the system. From now on we will mainly deal with a *request sequence* $\omega = \big((r_1, t_1), \ldots, (r_m, t_m)\big)$ instead of a request set, that is requests are ordered according to their release times. Note that m is used to denote the length of the request sequence.

The task of an *online algorithm for* Darp is to determine a sequence of schedules specifying the order of the requests and the paths between them. (Since we always deal with trees paths between requests are unique and are thus not strictly necessary.) The goal is to minimize an objective function; we restrict ourselves to the overall completion time and the total travel distance. Notice that in the offline version minimizing the completion time is equivalent to minimizing the total distance traveled, which is the objective of offline Darp.

We assume that the online algorithm does not know neither the total number of requests nor the last release time. Otherwise we could trivially be 1-competitive w. r. t. total travel distance: Just wait until the last request has arrived and serve all requests as the offline server does.

Definition 5.1 (Schedule) A *schedule* for a request set $A = \{r_1, \ldots, r_m\}$ is a sequence of time-request-pairs $\big((\tau_1, r_{i_1}), \ldots, (\tau_m, r_{i_m})\big)$ such that

- each request r_i occurs exactly once and
- request $r_{i_{k+1}}$ can be reached at time τ_{k+1} if request r_{i_k} is started at time τ_k.

Notice that waiting is allowed.

Definition 5.2 (Online Dial-a-Ride Problem ONLINEDARP)

ONLINEDARP

Instance: A graph $T = (V, E)$, a request sequence $\omega = \big((r_1, t_1), \ldots, (r_m, t_m)\big)$, a length function $d\colon E \to \mathbb{R}_{\geq 0}$ for the edges of G and a distinguished vertex $o \in V$.

Output: A *feasible* sequence of *current schedules* $(S_1, \ldots, S_m)$ such that the objective (overall completion time or total travel distance) is minimized. Feasibility means:

- The first current schedule S_1 starts at the depot o and the last current schedule S_m ends there.
- If the last current schedule S_m has been finished all requests have been served.
- Each current schedule S_i contains only requests known at time t_i (i. e., with release time $t_j \leq t_i$) and not already served by the schedules S_j, $1 \leq j < i$.
- When changing from current schedule S_i to S_{i+1}, a possibly interrupted request is immediately finished by S_{i+1} (non-preemptive transportation).

Observe that since we are considering arbitrarily long request sequences and an online algorithm has no way to determine when the last request arrives it is of no use to wait a fixed time in order to mimic the offline algorithm.

To extend our offline random model (the list random model) to online instances, we have to say how the release times are chosen, which can be done in essentially two ways. Either they are provided by a deterministic process or they are determined by a suitable random model. We will define a variant with deterministically chosen release times and one where a nice probability distribution is used.

Definition 5.3 (Online model 1) Let m be an integer. The random request sequence ω of the Online Model 1 is constructed by choosing a request list L_m according to a distribution $(p_v)_{v \in V}$ and the sequence of release times t_i, $1 \leq i \leq m$, in any deterministic way.

Definition 5.4 (Online model 2) For an integer m let $X_1, \ldots, X_m$ be random variables which are all exponentially distributed with parameter λ, i. e., their distribution function is given by

$$F_{X_i}(t) = \begin{cases} 0 & x < 0 \\ 1 - \mathrm{e}^{-\lambda t} & x \geq 0. \end{cases}$$

Then the request release times for a random request list chosen as in the list random model are the times T_i defined by

$$T_i = \sum_{j=1}^{i} X_j \qquad 1 \leq i \leq m.$$

5.2 Deterministic Competitiveness Results for OnlineDarp

The main theoretical tool for evaluating online algorithms is the so-called *competitive analysis*. The idea is to compare an online algorithm to an optimal offline algorithm knowing the entire online instance (request sequence in our case) in advance.

Let us review the requirements for an *online algorithm*: An online algorithm maintains a current schedule for a subset of all requests known so far, which is executed by the server. (The schedule may be empty indicating that the server does nothing.) The algorithm may update this schedule at arbitrary times but is restricted to use only information available at that time.

In contrast, an *offline algorithm* is provided with the entire request sequence ω at once, i. e., it can "anticipate the future". Especially interesting are the *optimal offline algorithms* which always give optimal solutions. We will denote an optimal offline algorithm for OnlineDarp by Opt. Clearly, Opt will always be at least as good as *any* online algorithm, since any online solution (i. e., sequence of schedules) can be converted to an offline schedule.

We say that an online algorithm OlAlg solves OnlineDarp (or is an algorithm for OnlineDarp) if its output is a feasible sequence of current schedules for every input instance. (Strictly speaking OlAlg would have to return an optimal solution.) We can now define our performance measure for online algorithms.

Definition 5.5 (Competitive ratio) Let OlAlg be any online algorithm for OnlineDarp. OlAlg is said to be *c-competitive for* OnlineDarp if for any (finite) request sequence ω there is a constant c satisfying

$$\textsc{OlAlg}(\omega) \leq c \cdot \textsc{Opt}(\omega).$$

The infimum of all c with this property is the *competitive ratio of* OlAlg and denoted by c_{OlAlg}.

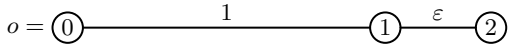
Observe that we do not say anything about the efficiency of OlAlg. The competitive ratio measures the impact of the lack of information on the performance of online algorithms with unrestricted computational power. Of course we would like to have efficient online algorithms. This is even more important since in online environments there are often real-time requirements, so decisions must be made fast.

The technique of determining c_{OlAlg} as well as providing lower bounds for *any* online algorithms is known as *competitive analysis*.

A very fertile view on this kind of analysis is to think of it as a game between the online algorithm and a *malicious adversary*. Since the online algorithm does not know of future requests, we can imagine our request sequence ω being built by some adversary, who successively responses to the choices of OlAlg, trying to make c_{OlAlg} as large as possible. Of course, since OlAlg is deterministic its choices can be computed in advance and the adversary can in principle choose the request sequence ω at once. The following two results show how to exploit this view in the construction of lower bounds for the competitive ratio. The first of them is probably folklore.

Theorem 5.6 *There is no competitive deterministic online algorithm for the* OnlineDarp *w. r. t. total travel distance.*

Proof. Fix an online algorithm OlAlg. Consider the following simple transportation network, where epsilon is a small positive constant less than 1.



We now construct a request sequence of length m. The first request $r_1 = (2, 1)$ is released at time 0. Algorithm OlAlg has to serve r_1 and will finally return to the depot $o = 1$. Assume this happens at time t_1. The second request $r_2 = (2, 1)$ is released at time t_1. Again, algorithm OlAlg will eventually return to the origin at some time t_2, at which a request $r_3 = (2, 1)$ is released. This process continues until m requests have been determined.

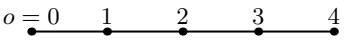
Clearly, OlAlg is forced to serve every request separately resulting in a total travel distance of $2m(1 + \varepsilon)$. The optimal offline solution is to travel to vertex 2, serve all m requests and to return to the origin. Thus the optimal travel distance is $2(1 + m\varepsilon)$. The competitive ratio is thus at least

$$\frac{2m(1+\varepsilon)}{2(1+m\varepsilon)} \overset{m\to\infty}{=} \infty.$$

□

Theorem 5.7 *[2] No deterministic algorithm for* OnlineDarp *can achieve a competitive ratio smaller than* $\frac{5}{3}$ *w. r. t. completion time.*

Proof. The instance used for this lower bound is the path of length 5, again with unit weights on the edges.



Note that the number of a vertex is its distance from the depot o.

Now assume that OlAlg is a deterministic c-competitive online algorithm with $c < \frac{5}{3}$.

At time $t = 0$, we issue two requests $r_1 = (0, 2)$ and $r_2 = (2, 0)$. Clearly, $\textsc{Opt}(r_1, r_2) = 4$ and thus OlAlg has to start serving request r_2 at some time T, $2 \leq T \leq 4c - 2$ in order to be c-competitive.

The remaining request sequence depends on T and we distinguish two cases.

Case 1: $2 \leq T \leq 3$

At time T the request $r_3 = (3, 2)$ becomes known. OlAlg has just started serving r_2 so it needs $\textsc{OlAlg}(r_1, r_2, r_3) = T+2+6 \geq 10$ time, whereas $\textsc{Opt}(r_1, r_2, r_3) = 6$. Thus c must be at least $\frac{5}{3}$.

Case 2: $3 < T < 4c - 2 \leq \frac{14}{3}$

The request $r_3 = (\lfloor T \rfloor, 2)$ is released at time T. The online algorithm has to serve r_2 first and thus needs time $\textsc{OlAlg}(r_1, r_2, r_3) \geq T + 2 + 2\lfloor T \rfloor$. On the other hand, the server controlled by the optimal offline algorithm has traveled to vertex $\lfloor T \rfloor$ and can serve r_3 and r_2 at once, requiring time $\textsc{Opt}(r_1, r_2, r_3) = T + \lfloor T \rfloor$. Since the function

$$f(T) := \frac{T + 2 + 2\lfloor T \rfloor}{T + \lfloor T \rfloor}$$

is monotonic decreasing in the intervals $(3, 4)$ and $[4, \frac{14}{3}]$ and we have

$$\lim_{T \to 4-} f(T) = \frac{12}{7} > \frac{5}{3} \quad \text{and} \quad f\left(\tfrac{14}{3}\right) = \frac{22}{13} > \frac{5}{3}$$

we can conclude $c > \frac{5}{3}$, which is a contradiction. □

5.3 Probabilistic Extensions: Randomized and Probabilistic Competitive Analysis

Although competitive analysis has been applied quite successfully, there are many interesting online problems for which it does not yield useful results. More precisely,

there are some online problems where it can be shown that *no* (deterministic) online algorithm can perform considerably better than a more or less trivial algorithm. This implies that competitive analysis fails in discriminating different online algorithms w. r. t. their performance, so we do not get any decision support which algorithm to use in practice. For example, both the strategies IGNORE and REPLAN discussed in the next section are $\frac{5}{2}$-competitive w. r. t. completion time when using optimal algorithms for reoptimization. (We have to admit that ONLINEDARP w. r. t. completion time is quite well-behaved since it allows constant competitive ratios. Typically, competitive ratios depend on some instance size parameter.)

This flaw of competitive analysis mainly stems from the adversary being too powerful: He can anticipate the behavior of any deterministic online algorithm and thus easily exploit the online algorithm's weaknesses. To overcome this problem several extensions were developed. The most common approach is to consider *randomized online algorithms*, i. e., algorithms whose decisions are not based on the input sequence alone, but also on random bits which are assumed to be independent of the input sequence. In effect this means that we are not doing worst-case analysis any longer since the performance of a randomized algorithm does not depend solely on the input, but the algorithm has a chance to perform better for some (hopefully many) choices of the random bits.

In defining the adversary for a randomized online algorithm ROLALG (and thus our yardstick for measuring its performance) we have to choose whether or not the adversary knows something about the random choices of ROLALG. This leads to the distinction between *oblivious* and *adaptive* adversaries.

Definition 5.8 (Adversaries for randomized online algorithms) Consider a randomized online algorithm ROLALG for a given online problem.

- The *oblivious adversary* OBL chooses the request sequence ω at once based on ROLALG (especially the probability distribution of ROLALG). The cost of OBL are the optimal offline cost of ω.

- An *adaptive adversary* builds a request sequence ω step-by-step in response to ROLALG's online decisions. There are two types of adaptive adversaries, differing in the cost they incur: The *adaptive offline adversary* ADOFF serves ω at the cost of an optimal solution, whereas the *adaptive online adversary* ADON decides how to serve a request as soon as it is generated and thus serves ω in an online fashion.

In the context of Darp, we can describe the difference between AdOn and AdOff more explicitly. AdOff builds a request sequence in interaction with RolAlg, depending on RolAlg decisions; finally, AdOff chooses a schedule for the requests which is used to determine the cost of AdOff (in our case the completion time or the total travel distance). In contrast, AdOn has to come up with a schedule for all yet unserved requests for every arriving request. It incurs cost in the same way as the online algorithm does, i. e., between successive requests, costs are determined according to the current schedule. Due to the unpredictable behavior of RolAlg, AdOn may have to do some additional empty rides, which AdOff certainly can avoid.

Notice that Obl is the weakest of these adversaries, because it has access to the least information. Similarly, AdOn is weaker than AdOff, since AdOn's solution can be used by AdOff and is thus at least as expensive as an optimal offline solution for that request sequence.

Definition 5.9 (Randomized competitive ratio) Let RolAlg be a randomized online algorithm and $\text{Adv} \in \{\text{Obl}, \text{AdOn}, \text{AdOff}\}$ be an adversary. The *randomized competitive ratio of* RolAlg is the infimum of all c satisfying

$$\mathbb{E}\left[\text{RolAlg}(\omega)\right] \leq c \cdot \mathbb{E}\left[\text{Adv}(\omega)\right],$$

denoted by $c^{\text{Adv}}_{\text{RolAlg}}$. RolAlg is said to be $c^{\text{Adv}}_{\text{RolAlg}}$-*competitive* if $c^{\text{Adv}}_{\text{RolAlg}} \leq \infty$.

When establishing a certain randomized competitive ratio of a randomized online algorithm the oblivious adversary is used most often. Randomized competitive analysis often improves the competitive ratio, for example from $\mathcal{O}(k)$ to $\mathcal{O}(\log k)$ for some instance size parameter k (this is the case for the paging problem, see [9]).

However, randomized competitive analysis is not of much use for deterministic algorithms which cannot be randomized easily, as it is the case for the algorithm use-MST considered so far. Similarly to the offline case, another way to replace worst-case analysis by some probabilistic method is to *randomize the input instead of the algorithm* and to do *probabilistic competitive analysis*, which is just an extension of offline probabilistic analysis to the online case.

The advantages of probabilistic analysis over worst-case analysis are the same as in the offline case: In reality, there is no malicious adversary and the orientation towards worst case instances may lead to algorithms which are inferior on actual instances. When doing probabilistic analysis, one has the possibility to model real-world instances by a suitable probability distribution (although an analysis for a more complex distribution may be intractable).

Definition 5.10 (Probabilistic competitive ratio) Let OLALG be a (deterministic) online algorithm and suppose that request sequences ω of length m are chosen according to a probability distribution, one for each $m \in \mathbb{N}$. The *probabilistic competitive ratio of* OLALG is the infimum of all c satisfying

$$\text{OLALG}(\omega) \leq c \cdot \text{OPT}(\omega) \quad \text{a. a. s.},$$

denoted by $c^{\text{prob}}_{\text{OLALG}}$. OLALG is said to be *a. a. s.-$c^{\text{prob}}_{\text{OLALG}}$-competitive* if $c^{\text{prob}}_{\text{OLALG}} \leq \infty$.

Remark There is another way to analyze the behaviour of online algorithms on randomized request sequences: *average-case competitive analysis*. When doing average-case analysis one is interested in *expected* behaviour instead of the behaviour of the algorithm on almost all request sequences.

There are two notions of average-case competitiveness used in the literature. The papers by Becchetti et al. [7] and by Fujiwara and Iwama [20] define the average-case competitive ratio of an online algorithm OLALG as

$$c^{\text{avg}}_{OlAlg} := \mathbb{E}_\omega \left[\frac{\text{OLALG}(\omega)}{\text{OPT}(\omega)} \right].$$

There is a slightly different definition in the scheduling literature ([35, 33]), which has also been studied by Becchetti et al. The average-case competitive ratio is then defined by

$$c^{\text{avg}}_{OlAlg} := \frac{\mathbb{E}_\omega \left[\text{OLALG}(\omega) \right]}{\mathbb{E}_\omega \left[\text{OPT}(\omega) \right]}.$$

However, our notion is much stronger (at least asymptotically) since we require that the competitive ratio is attained for almost all request sequences, whereas the expectation-based approach admits that the behaviour on a large part of the sequences is much worse, as long as there are enough instances which are handled substantially better.

5.4 The Strategies IGNORE and REPLAN

Ascheuer et al. [2] also give two natural strategies REPLAN and IGNORE for constructing online algorithms from offline algorithms. The idea is simple: Just compute at certain times an (optimal) schedule for the current request set. One then has to choose appropriate *reoptimization times* and what to do with the schedule from the last reoptimization. REPLAN and IGNORE are in a sense extreme in this respect: IGNORE reoptimizes only after the last schedule has been finished while REPLAN

tries to make use of further knowledge as soon as possible and thus discards the old schedule if a new one is computed. Note that both Ignore and Replan make no assumptions on the underlying network and thus can be used to obtain online algorithms for any graph (or metric).

Definition 5.11 (Ignore-strategy) Suppose that Alg is an algorithm for Darp. An online algorithm based on Alg works as follows:

- Initially the server is idle. As soon as requests arrive, we collect the requests and solve the associated Darp instance by employing Alg. The result is a schedule for the requests known so far.
- The server immediately starts executing this schedule. In the meantime, all newly released requests are collected to form the request set for the next reoptimization.
- If the server finishes its schedule and there are no further requests the server becomes idle. Otherwise, a new schedule is computed as described above and the server executes it.

This online algorithm will be called Ignore(Alg).

Definition 5.12 (Replan-strategy) Let Alg be any algorithm for the following slightly modified version of Darp: Additionally, we are given some point v in the continuous metric associated with Darp's graph G (see Definition A.2) and the goal is to find an optimal tour starting in v (instead of o) and ending in o.

The online algorithm Replan(Alg) works as follows:

- Initially the server is idle and located at the origin o. As soon as requests arrive, we solve a Darp instance arising from the requests known so far using Alg to obtain a schedule.
- The server immediately starts executing this schedule. If a new request is released, the server finishes its current transportation if there is one. Let v be the position of the server position. (Note that we need the continuous version of the graph metric here.) Compute a new schedule by invoking Alg on the current request set and v. The server now follows the new schedule.

Note that both online strategies Ignore and Replan are polynomial-time algorithms whenever the offline algorithm employed by them is a polynomial-time algorithm.

The interesting thing about REPLAN and IGNORE is that they can in fact be used to obtain competitive algorithms w. r. t. completion time.

Theorem 5.13 ([2],[23]) *Let* ALG *and* ALG′ *be ϱ-approximation algorithms for* DARP *and the modified version of* DARP *given in the definition of* REPLAN*, respectively. Then both* IGNORE(ALG) *and* REPLAN(ALG′) *are $\frac{5\varrho}{2}$-competitive w. r. t. completion time.*

Thus if we plug an optimal algorithm in IGNORE or REPLAN we get a $\frac{5}{2}$-competitive online algorithm.

Later in this chapter we will study the IGNORE-strategy in more detail, so let us introduce some terminology and notation. The time between IGNORE's successive reoptimizations will be called a *phase*. Clearly, IGNORE partitions the random request sequence ω in P subsequences $\omega_1, \ldots, \omega_P$ corresponding to the phases. Request subsequence ω_i consists of m_i requests and the server needs time Δ_i to complete the schedule computed by IGNORE.

Before turning to a probabilistic competitive analysis of IGNORE(USE-MST) for a special situation, we have a look at how the structural results for offline instances (Chapter 4) transfer to the "snapshot" problems solved by IGNORE and REPLAN.

5.5 Structure of Snapshot Problems

In this section we deal with the structure of the problems solved by IGNORE and REPLAN to obtain the new schedule, which will be called *snapshot problems* for short.

Proposition 5.14 *Suppose the* IGNORE*-strategy is used on a request sequence $\omega = \big((r_1, t_1), \cdots, (r_l, t_l)\big)$ which is obtained by choosing the requests r_i as in the list random model and the release times t_i in any fixed fashion (deterministically or probabilistically). Denote by $A(\omega_i)$ the set of requests to be served in phase i. Then the request sets $A(\omega_i)$ are distributed exactly as the random request sets of size m_i constructed by the list random model:*

$$\text{Prob}\left[A(\omega_i) = A \mid m_i = k\right] = \text{Prob}\left[A(L_k) = A\right].$$

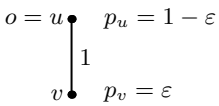
Proof. Observe that both $A(\omega_i)$ and $A(L_m)$ are actually constructed by building request lists and then "forgetting" the order of requests. It thus suffices to show that the distributions of the request lists coincide. In both cases the same distribution

(namely $(p_u \times p_v)_{(u,v)\in V\times V}$) is used to determine each request. The distributions on the request lists are just the product distribution and therefore equal. □

The situation is not as comfortable when the Replan-strategy is employed. The problem with Replan is that some requests might be in the system for a long time. This behavior is caused by the following situation: The server is located in a region where many requests occur and the request set features a request far away from that region. As long as there are new requests in this region the server may not travel towards the distant request, which is delayed longer and longer. That is a serious flaw of the Replan-strategy and it can even be proved that the maximum flow time is unbounded for deterministically chosen request sequences [24, 23]. In fact, this effect destroys the initial random structure of the requests.

Proposition 5.15 *Suppose the* Replan*-strategy is used on a request sequence* $\omega = \big((r_1,t_1),\cdots,(r_l,t_l)\big)$ *obtained by choosing the requests* r_i *as in the list random model and the release times* t_i *deterministically, i. e., by online model 1. Then the snapshot request sets are in general not compatible to any probability distribution on the vertices i. e., there is no probability distribution on the vertices allowing to describe the request distribution as in the list random model.*

Proof. Consider the following simple tree and corresponding probability distribution on the vertices ($\varepsilon < \frac{1}{2}$):



We now release $k+1$ requests $r_1,\ldots,r_{k+1}$ in the following fashion:

- The first k requests are released at time 0.
- The last request r_{k+1} is released at time δ, $0<\delta<1$.

Note that most of the k first requests will be u-loops and are thus served instantaneously. Let X_{uu}, X_{uv}, X_{vu} and X_{vv} be the number of requests along those arcs in the snapshot problem at time δ. Clearly, their expectations are

$$\begin{aligned}\mathbb{E}\left[X_{uu}\right] &= (1-\varepsilon)^2\\ \mathbb{E}\left[X_{uv}\right] = \mathbb{E}\left[X_{vu}\right] &= (k+1)\varepsilon(1-\varepsilon)\\ \mathbb{E}\left[X_{vv}\right] &= (k+1)\varepsilon^2,\end{aligned}$$

because only the u-loop-requests have been served so far. If the request set of this snapshot problem was distributed according to a distribution $(p'_v)_{v\in V}$ on the vertices as in the list random model $\mathbb{E}[X_{uu}]$ would be proportional to $p'_u \cdot p'_u$, $\mathbb{E}[X_{uv}]$ proportional to $p'_u \cdot p'_v$ and so on. However, since $\varepsilon < \frac{1}{2}$ we get from the last two equations that $(k+1)\varepsilon(1-\varepsilon) > (k+1)\varepsilon^2$, implying $p'_u > p'_v$. On the other hand, we have for sufficiently large k that $(k+1)\varepsilon^2 > (1-\varepsilon)^2$, implying $p'_v > p'_u$, which is a contradiction to the last implication. □

5.6 Probabilistic Analysis of the High Load Case

We are mainly interested in the long-term behavior of our online algorithms and therefore consider long request sequences. The completion time is not a sensible measure of performance for online algorithms in this setting, since the competitive ratio gets arbitrarily close to 1 by just increasing the request sequence length. For this reason, we will study the total travel distance as an alternative objective: An online algorithm performs well if it does not do many empty rides.

In this section we show that IGNORE(USE-MST) is a. a. s.-$(1+o(1))$-competitive w. r. t. the total travel distance if there is enough load. We need to require high load since otherwise the behaviour found in the deterministic lower bound for ONLINEDARP w. r. t. total travel distance will occur in the probabilistic setting, too, and we will not achieve competitiveness. Recall that this lower bound exploited the fact that the server has to return to its depot to enforce unavoidable empty rides for the online server. When working on trees the unavoidable empty rides are determined essentially by the balancing arcs.

We assume a fixed tree $T = (V, E)$ as underlying transportation network and that the requests are randomly generated independently of each other according to a vertex distribution $(p_v)_{v\in V}$ as in the list random model. Additionally we require that the expected request length given by μ is larger than zero and the variance σ^2 of the request length is finite. The corresponding release times are constructed by online model 2.

For shortness, we will use IGNORE instead of the more precise IGNORE(USE-MST) from now on. Intuitively, we will exploit that the number of requests per phase is high in a high load situation and the requests can be scheduled in a short tour due to synergy effects because the number (and length) of balancing arcs needed will be comparatively small.

We know from the discussion of the solution structure of DARP in Chapter 2 that

every solution has to use at least the request arcs and the balancing arcs, whereas suboptimal solutions use more linking arcs than necessary. We can exploit this knowledge to obtain good lower and upper bounds for the traveled distance of Opt and Ignore, respectively.

Denoting by $L(\omega)$ the total length of the requests in ω and by $B'(\omega)$ the length of unavoidable restricted balancing arcs for request sequence ω, we get the lower bound

$$\text{Opt}(\omega) \geq L(\omega) + B'(\omega) \tag{5.1}$$

for Opt. Similarly, we can bound Ignore(ω) as

$$\text{Ignore}(\omega) \leq L(\omega) + \sum_{i=1}^{P} B'(\omega_i) + 2P|E|, \tag{5.2}$$

where the last term is used as an upper bound for the total length of a MST per phase.

In this analysis we will exploit the fact that the snapshot problems solved by Ignore have the same stochastic structure as the offline problem which was established in Proposition 5.14.

We divide the set of phases into *epochs*: For a fixed (small) parameter $\varepsilon \in (0, 1)$, phase i belongs to epoch j ($j = 1, \ldots, \frac{1}{\varepsilon} + 1$) if

$$m^{(j-1)\varepsilon} \leq m_i < m^{j\varepsilon}.$$

We will require that starting in epoch 2 the number of requests per phase will be strictly monotonic increasing, which happens a. a. s. This justifies the name "epoch". The first epoch will be treated separately for technical reasons: In the first epoch, the number of requests in a phase does not depend on m, so we are not able to say anything about its asymptotic behaviour.

In order to arrive at the promised $(1 + o(1))$-competitive ratio, we will show that the following properties are satisfied a. a. s. if the load is sufficiently high:

1. The length L_m of m requests satisfies $L_m \geq (1 - o(1))\mu m$ (Proposition 5.17).

2. There are at most m^{ε} phases with at least a constant number of requests per phase in epoch 1 and at most $\mathcal{O}(\log m)$ phases in epochs j, $j \geq 2$ (Proposition 5.19).

3. There are at most $\mathcal{O}(\sqrt{m} \log m) \subseteq o(m)$ balancing arcs for m requests. Furthermore, there are at most $\mathcal{O}\left(\sqrt{m_i} \log m_i\right)$ balancing arcs generated by Ignore in phase i during epochs $2, \ldots, \frac{1}{\varepsilon} + 1$ (Proposition 5.22).

All these considerations are based on request sequences generated by Online Model 2 and T as underlying tree.

Theorem 5.16 *Let the tree $T = (V, E)$ and the edge length function $d\colon E \to \mathbb{R}_{\geq 0}$ as well as the vertex probability distribution $(p_v)_{v \in V}$ be such that for the expectation μ and variance σ^2 of the length of a single request we have that $\mu > 0$ and $\sigma^2 < \infty$.*

IGNORE(USE-MST) *working on the tree T with request sequences provided by online model 2 according to $(p_v)_{v \in V}$ achieves a probabilistic competitive ratio (w. r. t. total travel distance) of $1 + o(1)$ as $m \to \infty$ if the arrival rate λ satisfies $\lambda = \frac{1}{\mu} + \delta$ for some constant $\delta > 0$.*

Proof. We tacitly assume that the properties 1, 2 and 3 stated above hold.

We first need a good bound on the length of balancing arcs produced by IGNORE, given by the term $\sum_{i=1}^{P} B'(\omega_i)$ in Equation (5.2). The total number of balancing arcs for the at most m^ε phases in epoch 1 is maximized if each phase contains only a single request. To see this, let k be the number of requests in those m^ε phases and fix a constant C such that the function $f(x) := C\sqrt{x} \log x$ gives an upper bound $f(m)$ on the number of balancing arcs for m requests. Note that f is concave and there are not more than k phases. The question is now for which choice of $m_1, \ldots, m_k$ the sum

$$\sum\nolimits_{m_1,\ldots,m_k\colon \sum_i m_i = k} f(m_i)$$

is maximized. Applying Jensen's inequality (see Theorem A.8) to $\theta_i = \frac{1}{k}$ and real variables x_i subject to $\sum_i x_i = k$ we get

$$f\left(\sum\nolimits_{i=1}^{k} \tfrac{1}{k} x_i\right) = f(1) \geq \sum\nolimits_{i=1}^{k} \tfrac{1}{k} f(x_i).$$

If we multiply by k our claim follows. We have just seen that the length of balancing arcs in the first epoch is in $\mathcal{O}(m^\varepsilon)$ since the length of each balancing arc is bounded by a constant.

Now let us estimate the number of balancing arcs Z generated in later epochs. By definition, every phase i in epoch j serves at most $m^{(j+1)\varepsilon}$ requests. As there are at most $\mathcal{O}(\log m)$ phases per epoch, we have

$$\begin{aligned} Z &\leq \mathcal{O}(\log m) \sum\nolimits_{j=2}^{1/\varepsilon+1} \mathcal{O}\left(m^{(j+1)\varepsilon/2} \log m_i\right) \\ &\leq \mathcal{O}\left(\log^2 m\right) \sum\nolimits_{j=2}^{1/\varepsilon+1} m^{(j+1)\varepsilon/2} \\ &\leq \mathcal{O}\left(\log^2 m\right) \int_{x=2}^{1/\varepsilon+2} m^{(x+1)\varepsilon/2}\,\mathrm{d}x \end{aligned}$$

$$= \mathcal{O}\left(\log^2 m\right) \frac{2}{\varepsilon \ln m} m^{3\varepsilon/2}(\sqrt{m}-1)$$
$$\leq \mathcal{O}\left(m^{(1+3\varepsilon)/2} \log m\right).$$

Again, the length of a balancing arc is bounded by a constant so the total length of the balancing arcs is $\mathcal{O}\left(m^{(1+3\varepsilon)/2} \log m\right)$.

Using the last results in the lower and upper bounds provided by Equations (5.1) and (5.2) yields

$$\frac{\textsc{Ignore}(\omega)}{\textsc{Opt}(\omega)} \leq \frac{L(\omega) + \left(\mathcal{O}\left(m^{\varepsilon}\right) + m^{(1+3\varepsilon)/2} \log m\right) + \mathcal{O}\left(m^{\varepsilon}\right)}{L(\omega)}$$
$$\leq 1 + \frac{\mathcal{O}\left(m^{\varepsilon} + m^{(1+3\varepsilon)/2} \log m\right)}{c_\mu m}$$

where c_μ is some positive constant satisfying $L(\omega) \geq c_\mu m$ a. a. s. We see that this ratio is

$$= 1 + o(1) \quad \text{as } m \to \infty,$$

if we choose an $\varepsilon < \frac{1}{5}$, for instance $\varepsilon = \frac{1}{10}$.

Note that we did not use the length of balancing arcs in the lower bound for Opt. The main reason for this is that we simply do not have a lower bound for this. Since we have an upper bound of $\mathcal{O}\left(\sqrt{m} \log m\right)$ they would not contribute much anyway.□

A short discussion of this result is in order. First note that the requirement that requests arrive faster than they can be coped with is not a suitable assumption for real-world systems. In fact we do not have reasonable load (see [24]); queuing theorists call such a system *unstable* since the number of requests is ever-increasing if the request sequence continues infinitely.

The real story is that we wanted to know where the approach described by the lower and upper bounds (5.1) and (5.2) leads to. It turned out that the "unreasonable load" requirement is sufficient to prove a $(1 + o(1))$-competitiveness result. We do not know whether this requirement is necessary and it may well be that a similar asymptotic result may be obtained without resorting to probabilistic tools at all.

However, the result is interesting in its own right. It affirms the intuition that if there are many requests there will be synergy effects which can be exploited. Another issue is that the offline algorithm does not have a real advantage over the Ignore strategy. This is obvious if the constant δ is very large since then most requests arrive in a short interval and Ignore works essentially as the offline algorithm. But

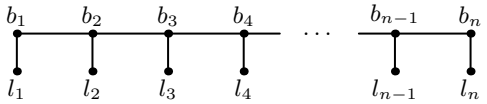
it is also true if δ is very small and there is (at least at the beginning) real online behaviour.

Finally, observe that this result applies also to IGNORE(OPT) since the upper bound for IGNORE(USE-MST) is naturally one for IGNORE(OPT). This implies that asymptotically IGNORE(USE-MST) is as good as IGNORE(OPT).

5.6.1 A Simple Special Case and Some First Observations

We first have a look at how a typical example of a tree and vertex distribution might look like. More specifically, we look at the so-called caterpillar graph, which has been the starting point for the probabilistic analysis presented in Chapter 4 (see [13]).

Example The *caterpillar graph* Cat_n (caterpillar for short) is the tree with n leaves $l_1, \ldots, l_n$, whose non-leaf vertices $b_1, \ldots, b_n$ make up a path of length $n-1$, called *backbone*. The edges $\{l_i, b_i\}$ are called *feet*.



Note that the caterpillar is just the underlying network of an elevator system: The backbone vertices b_i correspond to the floors and the weights of the feet can be used to model delays. According to this interpretation, requests enter and leave the system through the leaves l_i, so we will use a special random model for them. Furthermore, we consider only the caterpillar with uniform edge weights.

In the *Caterpillar random model* we assume that each request is of the form (l_i, l_j) and that all requests are equally probable, that is we choose the uniform distribution on the leaves for generating random requests.

Denote by L the random variable giving the length of a randomly chosen request. Notice that in Cat_n, no request with non-zero length can be shorter than 3 or longer than $n+1$. There are exactly $2(n-l+2)$ distinct requests of length l, namely twice the number of pairs (l_i, l_{i+l-2}). We thus have

$$\mathrm{Prob}\,[L=l] = 2\frac{n-l+2}{n^2} \quad \text{for } 3 \le l \le n+1.$$

Computing the expected request length yields

$$\begin{aligned}\mu = \mathbb{E}\,[L] &= \sum\nolimits_{l=3}^{n+1} l\ \mathrm{Prob}\,[L=l] \\ &= \frac{2}{n^2}\sum\nolimits_{l=3}^{n+1} l(n-l+2) \\ &= \frac{1}{3}\frac{n^2+6n-7}{n} = \tfrac{1}{3}n + \mathcal{O}\,(1)\,.\end{aligned}$$

Similarly, we get the variance

$$\begin{aligned}\sigma^2 = \operatorname{Var}[L] &= \mathbb{E}\left[L^2\right] - (\mathbb{E}[L])^2 \\ &= \frac{2}{n^2}\sum_{l=3}^{n+1} l^2(n-l+2) - \left(\frac{1}{3}\frac{n^2+6n-7}{n}\right)^2 \\ &= \frac{1}{18}\frac{n^4+25n^2+72n-98}{n^2} = \tfrac{1}{18}n^2 + \mathcal{O}(1).\end{aligned}$$ ■

Now let us return to the general case.

Proposition 5.17 *Let the tree $T = (V, E)$ and the edge length function $d\colon E \to \mathbb{R}_{\geq 0}$ as well as the vertex probability distribution $(p_v)_{v\in V}$ be such that for the expectation μ and variance σ^2 of the length of a single request we have that $\mu > 0$ and $\sigma^2 < \infty$.*

Let L_m be the total length of m requests. We have a. a. s. that

$$L_m \geq (1 - o(1))\mu m. \tag{5.3}$$

Proof. Clearly, $\mathbb{E}[L_m] = m\mu$. Applying Chebyshev's inequality, we see that

$$\operatorname{Prob}\left[|L_m - m\mu| \geq m^{2/3}\right] \leq \frac{m\sigma^2}{m^{4/3}} = \sigma^2 m^{-1/3} \in o(1) \quad \text{as } m \to \infty.$$ □

5.6.2 There Are Many Requests Per Phase

A very important issue in this analysis is the fact that there are not too many phases, because this keeps the total length of balancing arcs generated by Ignore small. To prove this fact we will need that the release rate is *larger* than the rate at which requests can be finished. In that case the number of requests should grow (on average) from phase to phase. There are two points to consider:

1. Show that once there are "lots of requests" the probability is very high that this property is also satisfied in the next phase.

2. Show that it will not take too many phases until there are the first time "lots of requests".

We will start by examining the first issue. The following lemma bounds the probability that in an interval of length Δ_i significantly less requests arrive than are expected. Recall that the number of requests generated by a process with exponentially distributed interarrival times is Poisson-distributed (see Section A.5.1).

Lemma 5.18 *Recall that m_i denotes the number of requests in phase i and that Δ_i is the time needed to serve these requests. Fix an $\alpha \in (0,1)$.*

$$\text{Prob}\left[m_{i+1} \leq \alpha\lambda\Delta_i \mid m_i = k\right] \in \mathcal{O}\left(\mathrm{e}^{-\lambda\Delta_i}\right).$$

Note that Δ_i implicitly depends on k.

Proof. Using the distribution function of the Poisson distribution we can express the left hand side as

$$\begin{aligned}\text{Prob}\left[m_{i+1} \leq \alpha\lambda\Delta_i \mid m_i = k\right] &\leq \sum\nolimits_{i=0}^{\alpha\lambda\Delta_i} \frac{(\lambda\Delta_i)^i}{i!}\mathrm{e}^{-\lambda\Delta_i} \\ &= \mathrm{e}^{-\lambda\Delta_i}\sum\nolimits_{i=0}^{\alpha\lambda\Delta_i}\frac{(\lambda\Delta_i)^i}{i!}.\end{aligned}$$

To prove the lemma we need to show that the sum at the right hand side is in $\mathcal{O}\left(\mathrm{e}^{c\lambda\Delta_i}\right)$ for some $c < 1$. To simplify our computations let $x := \lambda\Delta_i$ and $i_0 := \alpha x$. Since $\alpha < 1$ we know that the summands are increasing which leads to the bound

$$\begin{aligned}\sum\nolimits_{i=0}^{i_0}\frac{x^i}{i!} &\leq i_0\left(\frac{x^{i_0}}{i_0!}\right) + 1 \\ &\sim i_0\frac{x^{i_0}\mathrm{e}^{i_0}}{\sqrt{2\pi i_0}\,(\alpha x)^{i_0}} \\ &= \sqrt{\frac{i_0}{2\pi}}\left(\frac{\mathrm{e}}{\alpha}\right)^{i_0} \\ &= \sqrt{\frac{i_0}{2\pi}}\mathrm{e}^{i_0(1-\ln\alpha)} \\ &\leq \mathrm{e}^{\alpha(1-\ln\alpha)x+\mathcal{O}(1)} \in \mathcal{O}\left(\mathrm{e}^{cx}\right).\end{aligned}$$

As the function $f(\alpha) := \alpha(1-\ln\alpha)$ is strictly increasing in $(0,1)$ and $\lim_{\alpha\nearrow 1} f(\alpha) = 1$ the constant c is smaller than 1. □

Proposition 5.19 *Let the arrival rate be $\lambda = \frac{1}{\mu} + \delta$ for some (possibly small) positive constant δ and fix some $\varepsilon \in (0,1)$.*

Then the first epoch a. a. s. consists of at most m^ε phases with at least a constant number of requests per phase. Furthermore, in each epoch j, $j > 2$, the number of requests increases from phase to phase (with the possible exception of the last phase) and thus the epoch consists of $\mathcal{O}(\log m)$ phases a. a. s.

Proof. Let us first examine how many requests are unserved at time $t(m)$. Denote by $N_{t(m)}$ the number of requests already released at time $t(m)$ and by $D_{t(m)}$ the

number of requests already served. Clearly, $N_{t(m)}$ is Poisson-distributed with parameter $\lambda t(m)$, so we have $\mathbb{E}\left[N_{t(m)}\right] = \lambda t(m)$. Invoking Chebyshev's inequality with $t := \sqrt{t(m)}\log m$ tells us that $N_{t(m)} \geq (1 - o(1))\lambda t(m)$ a. a. s.

We know from Section 5.6.1 that k requests need a. a. s. at least time $(1 - o(1))k\mu$ to be served. If $t(m)$ is sufficiently large we can assume that each request takes time $(1 - o(1))\mu$, providing us with the bound $D_{t(m)} \leq \frac{t(m)}{(1-o(1))\mu} = \frac{(1+o(1))}{\mu}t(m)$. Combining both estimates we see that at time $t(m)$ there are at least

$$\begin{aligned} N_{t(m)} - D_{t(m)} &\geq (1 - o(1))\lambda t(m) - \frac{(1 + o(1))}{\mu}t(m) \\ &= t(m)\left((1 - o(1))\left(\frac{1}{\mu} + \delta\right) - \frac{(1 + o(1))}{\mu}\right) \\ &= t(m)\left((1 - o(1))\delta - \frac{o(1)}{\mu}\right) \\ &= \delta(1 - o(1))t(m) \end{aligned}$$

requests in the system. Choosing $t(m) := m^\varepsilon$ we have that after at most m^ε phases with at least constantly many requests there is a first phase with $\Omega(m^\varepsilon)$ requests.

It remains to establish that once Ignore-phases feature m^ε requests (i. e., as of epoch 2) the sequence of requests in the phases is strictly monotonic increasing a. a. s., that is $m_{i+1} > m_i$ for all phases $i + 1 \neq P$ in epoch j, $j \geq 2$. Assume for the moment that there are at most $\mathcal{O}(\log m)$ phases in those epochs and that the request generation process is not stopped after the mth request.

Suppose phase i_0 is the first phase of epoch 2. Since the length of k requests is a. a. s. $\geq (1 - o(1))\mu k$, we see that $\Delta_{i_0} \geq (1 - \varepsilon_0)\mu m_{i_0}$ a. a. s. for some suitable $\varepsilon_0 \in (0, 1)$. We want to invoke Lemma 5.18 to guarantee that $m_{i_0+1} > m_{i_0}$ a. a. s., so we need an α such that $\alpha\lambda\Delta_{i_0} > m_{i_0}$, or equivalently

$$\alpha\left(\frac{1}{\mu} + \delta\right)(1 - \varepsilon_0)\mu m_{i_0} > m_{i_0}.$$

From the last inequality we get the condition

$$\alpha > \frac{1}{(1 + \delta\mu)(1 - \varepsilon_0)}$$

for an appropriate choice of α. On the other hand, α has to be less than 1, leading to another condition on ε_0

$$1 > \frac{1}{(1 + \delta\mu)(1 - \varepsilon_0)} \quad \Longleftrightarrow \quad 1 - \varepsilon_0 > \frac{1}{1 + \delta\mu}.$$

There are values for α and ε_0 which obey all conditions so we get that $m_{i_0+1} > m_{i_0}$ a. a. s.

In the preceding argument we did not explicitly use that we dealt with the *first* phase of epochs $2, \ldots, \frac{1}{\varepsilon} + 1$, so this choice of α and ε_0 indeed guarantees $m_{i+1} > m_i$ for all $i + 1 > i_0$. We still have to consider the probability that we have this kind of "success" for *all* phases. By Bernoulli's inequality, the probability that $m_{i+1} > m_i$ for all $i \geq i_0$ is at least

$$\left(1 - \mathcal{O}\left(\mathrm{e}^{-\lambda m^{\varepsilon}}\right)\right)^{\mathcal{O}(\log m)} \geq 1 - \mathcal{O}(\log m) \cdot \mathcal{O}\left(\mathrm{e}^{-\lambda m^{\varepsilon}}\right) = 1 - o(1) \quad \text{as } m \to \infty.$$

Similarly, the probability that $\Delta_i \geq (1 - \varepsilon_0)\mu m_i$ for all $i \geq i_0$ is at least

$$\left(1 - \sigma^2 m^{-\varepsilon/3}\right)^{\mathcal{O}(\log m)} \geq 1 - \mathcal{O}(\log m) \cdot \sigma^2 m^{-\varepsilon/3} = 1 - o(1) \quad \text{as } m \to \infty,$$

where we made use of Proposition 5.17.

So far we have seen that the number of requests per phase is a. a. s. strictly monotonic increasing provided that there are at most $\mathcal{O}(\log m)$ phases in the last epochs and there are infinitely many requests. The number of requests per phase grows by a factor of $\alpha\lambda(1 - \varepsilon_0)\mu > 1$ so there are at most $\mathcal{O}(\log m)$ phases in an epoch. This holds also if we consider only the first m requests of the infinitely many requests, since then only the last phase P may violate the increasing-condition. Since there are not more than $\frac{1}{\varepsilon} + 1$ epochs the number of phases is indeed $\mathcal{O}(\log m)$ in total.□

5.6.3 Estimating the Number of Balancing Arcs

In the deterministic case (see Chapter 2) we used the variables $\Phi(u, v)$ and $\Phi(v, u)$ to count the number of request arcs using edge $\{u, v\}$ in the corresponding direction. The difference $\Phi(v, u) - \Phi(u, v) =: U(u, v)$ told us how many balancing arcs (u, v) we had to add. We are not interested in the direction, but only in the number of balancing arcs, so $|U(u, v)|$ is just the number of times edge $\{u, v\}$ is traversed by the balancing arcs. The total number Z of balancing arcs is

$$Z = \sum\nolimits_{\{u,v\} \in E} |U(u, v)|.$$

What happens to $U(u, v)$ if a new request is added? The first possibility is that this new request does not use edge $\{u, v\}$ so $U(u, v)$ does not change. Otherwise, $U(u, v)$ increases (decreases) by one if the request traverses $\{u, v\}$ from v to u (from u

to v). Thus we can express $U(u,v)$ as

$$U(u,v) = \sum_{i=1}^{m} X_i(u,v) \quad \text{for} \tag{5.4}$$

$$X_i(u,v) := \begin{cases} 1 & \text{request } i \text{ traverses } \{u,v\} \text{ from } u \text{ to } v \\ -1 & \text{request } i \text{ traverses } \{u,v\} \text{ from } v \text{ to } u \\ 0 & \text{else.} \end{cases} \tag{5.5}$$

All $X_i(u,v)$ are identically distributed. If we define

$$p_{uv} := \left(\sum\nolimits_{u' \in V(u)} p_u\right) \cdot \left(\sum\nolimits_{v' \in V(v)} p_v\right)$$

for each edge $\{u,v\} \in E$ the probability distribution of $X_i(u,v)$ is given by

$$\text{Prob}\left[X_i(u,v) = k\right] = \begin{cases} p_{vu} & k = 1 \\ p_{uv} & k = -1 \\ 1 - 2p_{uv} & k = 0. \end{cases}$$

Obviously, $U(u,v)$ behaves similar to the symmetric random walk with the difference that there is some positive probability of no change. We want to transfer the result on the expected distance for the symmetric random walk to this more general case. To this end, we give an equivalent random experiment for generating the vector $\big(X_i(u,v)\big)_{1 \le i \le m}$ which makes the connection explicit. Let $p := p_{uv} = p_{vu}$ and $q := 1 - 2p$.

Definition 5.20 (Equivalent random model for $U(u,v)$) Construct a random vector $x' = \big(X'_i(u,v)\big)_{1 \le i \le m} \in \{-1,0,1\}^m$ as follows:

- Choose a set $J \subseteq \{1,\ldots,m\}$ of size $|J|$ at random, where $|J|$ is binomially distributed:
$$\text{Prob}\left[|J| = j\right] = \binom{m}{j} q^j (2p)^{m-j}.$$
The set J determines the zeroes of x': $X'_j(u,v) := 0$ for all $j \in J$.

- Choose a realization $y = (Y_j)_{j \in \{1,\ldots,m-|J|\}}$ of the symmetric random walk of length $m - J$ at random. This determines whether the remaining positions are -1 or 1: Suppose that the non-zero positions are renumbered from 1 to $m - J$. Now define $X'_j(u,v) := Y_j$ for all $j \in \{1,\ldots,m-|J|\}$.

Lemma 5.21 *Fix an integer m. The distributions induced by a random walk according to Equation (5.4) and by the modified random model of Definition 5.20 on the set $x \in \{-1,0,1\}^m$ coincide.*

Proof. This proof is just a straightforward computation. Let $x \in \{-1,0,1\}^m$ and j be the number of zeroes in x.

$$\begin{aligned}\text{Prob}\left[\big(X_i(u,v)\big)_{1\le i\le m} = x\right] &= q^j p^{m-j}\\ \text{Prob}\left[\big(X'_i(u,v)\big)_{1\le i\le m} = x\right] &= \binom{m}{j} q^j (2p)^{m-j} \cdot \binom{m}{j}^{-1} \cdot \left(\frac{1}{2}\right)^{m-j}\\ &= q^j p^{m-j}.\end{aligned}$$

□

We can now directly exploit this fact to estimate $\mathbb{E}[Z]$ via $\mathbb{E}[|U(u,v)|]$. Intuitively it is clear that the last value should be smaller than the expected travel distance of a symmetric random walk of length m.

Proposition 5.22 *We have for the number of balancing arcs Z for m requests*

$$Z \le (1+o(1))|E|\sqrt{\frac{2m}{\pi}} \in \mathcal{O}\left(\sqrt{m}\log m\right) \quad \text{a. a. s.}$$

Starting from epoch 2, every request subsequence ω_i corresponding to phase i a. a. s. fulfills $|B'(\omega_i)| \in \mathcal{O}\left(\sqrt{m_i}\log m_i\right)$.

Proof. We compute $\mathbb{E}[|U(u,v)|]$ by using the modified random model for $U(u,v)$. Recall that W_k is just the symmetric random walk of length k (see Section A.5.2).

$$\begin{aligned}\mathbb{E}[|U(u,v)|] &= \sum_{j=0}^{m}\left\{\binom{m}{j} q^j (2p)^{m-j} \sum_{S\in\binom{\{1,\ldots,m\}}{j}} \binom{m}{j}^{-1} \mathbb{E}[|W_{m-j}|]\right\}\\ &= \sum_{j=0}^{m}\left\{\binom{m}{j} q^j (2p)^{m-j} \mathbb{E}[|W_{m-j}|]\right\}\\ &\le \mathbb{E}[|W_m|]\sum_{j=0}^{m}\binom{m}{j} q^j (2p)^{m-j}\\ &= \mathbb{E}[|W_m|] \sim \sqrt{\frac{2m}{\pi}}.\end{aligned}$$

From this we get the estimate

$$\mathbb{E}[Z] \le |E|\sqrt{\frac{2m}{\pi}} \in \mathcal{O}\left(\sqrt{m}\right).$$

To obtain the sharp concentration result, we apply Azuma's inequality (Theorem 3.11). In order to do this, we have to show that the function $f(r_1, \ldots, r_m) := \sum_{\{u,v\} \in E} |U(u,v)|$ satisfies the required Lipschitz-condition. Suppose that request r_i is replaced by request r_i'. How does this influence $U(u,v)$? We have already seen that $U(u,v)$ changes by at most 1 if a request is added and the same holds if a request is deleted. Thus, $|U(u,v)|$ increases by at most 2, so we get

$$|f(r_1, \ldots, r_{i-1}, r_i, r_{i+1}, \ldots, r_m)| - |f(r_1, \ldots, r_{i-1}, r_i', r_{i+1}, \ldots, r_m)| \leq 2|E|.$$

Substituting in Equation (3.8) yields

$$\begin{aligned} \text{Prob}\left[Z \geq \mathbb{E}[Z] + t\right] &\leq \exp\left(-\frac{t^2}{2\sum_1^m 4|E|^2}\right) \\ &= \exp\left(-\frac{t^2}{8|E|^2 m}\right). \end{aligned}$$

Choosing $t := \sqrt{m} \ln m$ shows that $Z \leq (1 + o(1))\mathbb{E}[Z] \ln m \in \mathcal{O}\left(\sqrt{m} \log m\right)$ a. a. s., namely with probability $\geq 1 - \frac{1}{m^2}$.

Consider the phases in epochs j, $j \geq 2$. In all of these phases there are at least m^ε requests and Proposition 5.19 states that there are at most $\mathcal{O}(\log m)$ of them. Thus the the probability that none of them violates the condition $|B'(\omega_i)| \in \mathcal{O}\left(\sqrt{m_i} \log m_i\right)$ is at least

$$\left(1 - \frac{1}{m^{2\varepsilon}}\right)^{\mathcal{O}(\log m)} \geq 1 - \mathcal{O}(\log m) \cdot \frac{1}{m^{2\varepsilon}} = 1 - o(1) \quad \text{as } m \to \infty.$$

by Bernoulli's inequality (see Theorem A.7). □

6 Conclusions and Outlook

In this thesis we studied an elementary *Dial-a-Ride problem* originally motivated by an elevator system from a theoretical point of view. Although this problem is drastically simplified compared to the real-world problem it is known to be NP-hard. We studied standard ways to deal with NP-hard optimization problems, namely *approximation algorithms* and *probabilistic analysis*, on this particular Dial-a-Ride problem. We also had a first look at an probabilistic online version of this problem.

We started by providing a graph-theoretic model called DARP for our special Dial-a-Ride problem. All of our results rely on the further assumption that the graph used here is a tree. Based on this model we described the $\frac{4}{3}$-approximation algorithm USE-MST running in nearly-linear time, which was proposed by Frederickson and Guan [18]. This algorithm has the important property that it is optimal whenever a certain associated graph constitutes a *star metric*.

It has been observed in computer experiments that USE-MST generates optimal solutions very frequently. This triggered the probabilistic analysis by Coja-Oghlan et al. [12] which we reviewed in detail in Chapter 4. This technical result was based on a suitably chosen random model which nevertheless entails a large class of request probability distributions. The main idea of this result is that almost all large instances are such that the associated graph mentioned above is indeed a star metric. At this point it payed off to know something about this special case, which is a further motivation to look at special cases of hard optimization problems. Another interesting thing about this analysis is its delicate interplay of combinatorial and probabilistic arguments.

The final chapter was devoted to first steps of a probabilistic analysis of the corresponding online problem. To this end we extended the classical competitive analysis to probabilistic competitive analysis. We were able to exploit the structure of solutions for DARP, which was already used in the analysis of USE-MST, in the online setting. We showed that the IGNORE strategy of Ascheuer et al. [1] employing USE-MST does achieve an asymptotically optimal probabilistic competitive ratio, i. e., is as good as an optimal offline algorithm, if requests arrive at a larger rate than can

be served.

We think there are the following interesting directions for further research:

- How do approximation algorithms for general graphs behave in practice and / or on random instances? Do they exhibit a better solution quality than their worst case approximation ratio? If so, why is this the case?
- Are there ways to exploit the results obtained for trees in the case of general graphs?
- Is the "unreasonable" load requirement necessary to obtain a constant probabilistic competitive ratio for IGNORE(USE-MST)? How does the performance of IGNORE(USE-MST) change if requests arrive less frequently?
- Are there lower bounds for the deterministic competitive ratio in high load situations?
- What can be said about the flow time? We think it is promising to look for related results in scheduling and queuing theory and to check how these can be adapted to Dial-a-Ride problems and the IGNORE and REPLAN strategies.
- How do the assumed probability distributions match the request distributions in practice? Can these probabilistic results be applied to the design of real-world systems?

A Technical Preliminaries

One challenging thing concerning the analysis of algorithms is the use of tools from many branches of mathematics and computer science. This appendix collects basic notions and facts used in this thesis.

A.1 Graph Theory

A.1.1 Basic Notions

As usual, a *graph G=(V,E)* is a pair of a *vertex set* V and an *edge (multi)set* E, each edge being an unordered pair of vertices, denoted by $\{u, v\}$. Similarly, a *directed graph* $G = (V, A)$ (or *digraph* for short) is a pair of a vertex set V and an *arc (multi)set* A, but now an arc (u, v) is an ordered pair of vertices.

For a graph G, we use the notation $\delta_G(v)$ for the *degree* of v, i. e., the number of edges incident to v. Analogously, we have the indegree $\delta_G^-(v)$ and outdegree $\delta_G^+(v)$ for vertices of a digraph G. We extend both $\delta_G^-(\cdot)$ and $\delta_G^+(\cdot)$ to subsets $U \subseteq V$ by taking the sum over all elements of U: $\delta_G^-(U) := \sum_{v \in U} \delta_G^-(v)$ and $\delta_G^+(U) := \sum_{v \in U} \delta_G^+(v)$.

In a digraph G, the set $\mathrm{Succ}(v)$ is the set of all direct successor vertices of v, that is

$$\mathrm{Succ}(v) := \{u \in V \mid (v, u) \in A\}.$$

A *walk* in a (di)graph G is a sequence of arcs (edges) $(u_1, v_1), \ldots, (u_l, v_l)$ such that $u_{i+1} = v_i$, $1 \leq i \leq l - 1$. If no vertex occurs more than once the walk is called a *path*. It is *closed* if additionally $v_l = u_1$ holds. A closed walk visiting each of its intermediate vertices only once is a *cycle*. An *Euler tour* is a closed walk visiting all arcs (edges) of the graph exactly once. A (di)graph is said to be *Eulerian* if it admits an Euler tour.

We say that an undirected graph is *connected* if for every pair of vertices $u, v \in V$ there is a path from u to v. For any undirected graph $G = (V, E)$ the binary relation $\sim$ defined on $V \times V$ by

$$u \sim v :\Longleftrightarrow \text{ there is a path from } u \text{ to } v$$

is an equivalence relation; the equivalence classes are called *connected components of* G. The symbol $\eta(G)$ gives the number of connected components of G.

In the case of digraphs there are two notions of connectedness: We say that a digraph $G = (V, A)$ is *weakly connected* if the *underlying undirected graph* $G' = (V, E)$ defined by

$$E := \{\{u, v\} \mid (u, v) \in A\}$$

is connected. Moreover, $G = (V, A)$ is said to be *strongly connected* whenever for every pair of vertices $u, v \in V$ there is a directed path from u to v. The *strongly connected components of* G are defined as in the undirected case.

A *tree* is a connected graph without any cycle. A tree can be turned into a *rooted tree* (which is a digraph) by choosing a vertex r as *root* and replacing all edges by arcs directed away from r. Such a rooted tree can be interpreted as a partial order $\preceq$; intuitively, we have $u \preceq v$ if and only if v lies on the (unique) path from r to u. We sometimes say that v *majorizes* u. Every non-root vertex v has a unique *parent vertex* $\varrho(v)$ determined by the arc $(\varrho(v), v)$.

If one removes an edge $\{u, v\}$ from a tree its vertex set decomposes into two connected sets $V(u)$ and $V(v)$ determined by $u \in V(u)$ and $v \in V(v)$. The edge $\{u, v\}$ is called a *cut edge* and the pair $(V(u), V(v))$ a *cut in* T.

Furthermore a graph $G' = (V', E')$ is said to be a *subgraph of* $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A subgraph G' of G is *spanning a vertex set* $U \subseteq V$ if it is connected and for every vertex $u \in U$ there is an edge in E' which is incident to u.

Often a graph is considered in connection with a non-negative distance function on the edges. We speak of a *weighted graph* in that case.

A.1.2 Graphs and Metrics

Many combinatorial optimization problems can be expressed in a natural way using metrics. A metric is simply an abstract way to "measure" distances.

Definition A.1 (Metric) A metric $(M, d\colon M \times M \to \mathbb{R}_{\geq 0})$ consists of a set of *points* and a function d giving distances between each pair of points. This distance function has to satisfy the following axioms:

1. $d(u, u) = 0$ for all $u \in M$
2. $d(u, v) = d(v, u)$ for all $u, v \in M$
3. $d(u, v) + d(v, w) \geq d(u, w)$ for all $u, v, w \in M$. (Triangle inequality)

Frequently there are only finitely many points. In that case a metric (M, d) can be interpreted as a weighted graph as follows: The vertices are just the points and we have an edge for any pair of points (including edges of the form $\{u, u\}$). The weight of each edge is defined to be the distance of the corresponding points. This compact representation is the reason why weighted graphs show up so prevalently in combinatorial optimization.

The converse, i. e., interpreting a weighted graph as a finite metric, is also possible. Consider a weighted graph $G = (V, E)$ with distance function $d\colon E \to \mathbb{R}_{\geq 0}$ and define by

$$D(u, v) := \text{length of a shortest } u\text{-}v\text{-path in } G \qquad \forall (u, v) \in V \times V$$

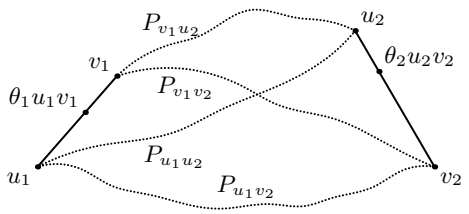
the *lifted distance function* $D\colon V \times V \to \mathbb{R}_{\geq 0}$. The lifted distance function D obeys the Triangle inequality due to this shortest path construction and the tuple (V, D) constitutes a metric known as *graph metric*.

For some applications it is useful to extend a graph metric to further points imagined to lie between the end points of an edge. The basic idea for this extension is that an edge $\{u, v\}$ is replaced by a copy of the interval $[0, 1]$. We need to remember the edge a point belongs to, so we "tag" it by "uv". To make the following definition unambiguous, we assume that the original points have been numbered $1, \ldots, n$.

Definition A.2 (Continuous graph metric) Assume a graph metric is given by a graph $G = (V = \{1, \ldots, n\}, E)$ and a distance function $d\colon E \to \mathbb{R}_{\geq 0}$. To construct the *continuous graph metric* (M_c, d_c) we first choose

$$M_c := \{\theta uv \mid \forall \{u, v\} \in E, u \leq v, \theta \in [0, 1]\}.$$

The intuition for defining the distance d_c between two points $\theta_1 u_1 v_1$ and $\theta_2 u_2 v_2$ is displayed in the following picture, in which $P_{\cdot,\cdot}$ is a shortest path in G between the indicated vertices:



Clearly, there are four possible paths between $\theta_1 u_1 v_1$ and $\theta_2 u_2 v_2$ so we set

$$d_c(\theta_1 u_1 v_1, \theta_2 u_2 v_2) := \text{length of the shortest of the paths}$$

- $\theta_1 u_1 v_1, u_1, P_{u_1 u_2}, u_2, \theta_2 u_2 v_2$
- $\theta_1 u_1 v_1, u_1, P_{u_1 v_2}, v_2, \theta_2 u_2 v_2$
- $\theta_1 u_1 v_1, v_1, P_{v_1 u_2}, u_2, \theta_2 u_2 v_2$
- $\theta_1 u_1 v_1, v_1, P_{v_1 v_2}, v_2, \theta_2 u_2 v_2$

where the length of $P_{\cdot,\cdot}$ is given by d and we additionally use the equations

$$d(\theta uv, u) := \theta d(u, v) \qquad d(\theta uv, v) := (1 - \theta) d(u, v).$$

A.1.3 Useful Facts

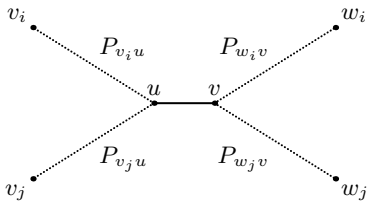
Lemma A.3 *[11] Let $G = (V, A)$ be a weakly connected digraph with at least one arc. The digraph G is Eulerian if and only if we have for any vertex $v \in V$*

$$\delta^+(v) = \delta^-(v). \qquad \square$$

Lemma A.4 (Pairing Lemma, [26]) *Let $T = (V, E)$ be a tree and U a subset of V with an even number of vertices. Then there is a pairing $(v_1, w_1), \ldots, (v_k, w_k)$ of the vertices in U such that the v_i-w_i-paths are all edge-disjoint.*

Proof. Suppose we know a pairing $(v_1, w_1), \ldots, (v_k, w_k)$ such that the total number of overlaps between the v_i-w_i-paths is minimal. We claim that in fact all v_i-w_i-paths are edge-disjoint in this pairing.

In order to derive a contradiction, assume that the paths P_i and P_j, connecting v_i to w_i and v_j to w_j respectively, share an edge $\{u, v\}$. Thus we have the following situation



and the paths $P_{\cdot,\cdot}$ are defined as depicted. We can now improve our pairing by replacing the pairs (v_i, w_i) and (v_j, w_j) by (v_i, v_j) and (w_i, w_j) which are connected by the paths $P_{v_i u} P_{uv_j}$ and $P_{w_i v} P_{v w_j}$, respectively. This operation decreases the number of overlaps by one, contradicting the minimality of our original pairing. $\square$

A.2 Asymptotics

The asymptotic notions and symbols used here are rather standard (see for example [25]).

We say that *two functions* $f, g\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ *are asymptotic to each other*, written $f \sim g$, if and only if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1.$$

Simple application of basic limit properties yields the laws

$$f \sim f', g \sim g' \implies f + g \sim f' + g', \quad f \cdot g \sim f' \cdot g', \quad \frac{f}{g} \sim \frac{f'}{g'}.$$

If f, g are such that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

we write this as $f \ll g$ or $f \in o(g)$. Similarly we write $f \in \mathcal{O}(g)$ and $g \in \Omega(f)$ if

$$\limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty.$$

The $\mathcal{O}(\cdot)$-Notation is usually used for giving upper bounds, whereas $\Omega(\cdot)$ denotes a lower bound. Furthermore we use $f \in \Theta(g)$ if we have both $f \in \mathcal{O}(g)$ and $f \in \Omega(g)$.

It is easy to see that we have the asymptotic hierarchy

$$c_1 \cdot 1 \ll c_2 \log n \ll c_3 n^{\varepsilon} \ll c_4 n^k \ll c_5 \mathrm{e}^n \ll c_6 n^n$$

for any constants $c_1, \ldots, c_6 \in \mathbb{R}_{>0}$.

A.3 Important Combinatorial Formulas

In the sequel we will make frequent use of binomial coefficients. Intuitively, the *binomial coefficient* $\binom{n}{k}$ gives the number of ways to choose k things out of n when the order does not matter, i. e., $\binom{n}{k}$ is the number of k-element subsets of a n-element set. By definition we have

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!} = \frac{(n)_k}{k!}.$$

We used so-called *falling factorials* on the right hand side: The symbol $(n)_k$ is defined as

$$(n)_k := n(n-1)\cdots(n-k+1).$$

Due to the already mentioned combinatorial interpretation we will use the notation $\binom{S}{k}$ to denote the set of all k-element subsets of a set S.

Put in a slightly different way, the binomial coefficient $\binom{n}{k}$ tells us how many different partitions of a n-element set into a k-element set and a $(n-k)$-element set exist. Sometimes we need to consider partitions in more than two sets. The number of distinct partitions of a n-element set into l subsets of size k_i, $1 \leq i \leq l$, is given by the *multinomial coefficient* (of course we assume $\sum_i k_i = n$)

$$\binom{n}{k_1, \ldots, k_l},$$

which can be expressed by binomial coefficients as

$$\binom{n}{k_1, \ldots, k_l} = \binom{n}{k_1} \cdot \binom{n-k_1}{k_2} \cdot \binom{n-k_1-k_2}{k_3} \cdots \binom{k_l}{k_l} = \frac{n!}{\prod_{i=1}^{l} k_i!}.$$

In order to estimate binomial coefficients we need to approximate the factorial function. This can be done by using the well-known *Stirling's formula* (see for example [21, 14]):

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{\mathrm{e}}\right)^n. \tag{A.1}$$

We will use it to estimate the binomial coefficient $\binom{2k}{k}$ as follows:

$$\begin{aligned} \binom{2k}{k} &= \frac{(2k)!}{k!\,k!} \\ &\sim \frac{\sqrt{4\pi k}\left(\frac{2k}{\mathrm{e}}\right)^{2k}}{\left(\sqrt{2\pi k}\left(\frac{k}{\mathrm{e}}\right)^k\right)^2} \\ &= \frac{2\sqrt{\pi k} \cdot (2k)^{2k} \cdot \mathrm{e}^{2k}}{2\pi k \cdot \mathrm{e}^{2k} \cdot k^{2k}} \\ &= \frac{2^{2k}}{\sqrt{\pi k}}. \end{aligned} \tag{A.2}$$

In fact, the right hand side of the last equation is actually an upper bound for $\binom{2k}{k}$, i. e., we have the bounds

$$\frac{1}{2}\frac{2^{2k}}{\sqrt{\pi k}} \leq \binom{2k}{k} \leq \frac{2^{2k}}{\sqrt{\pi k}}. \tag{A.3}$$

To see this, set $f(k) := \binom{2k}{k}$ and $g(k) := \frac{2^{2k}}{\sqrt{\pi k}}$. We will show that

$$\frac{f(k)}{g(k)} \leq \frac{f(k+1)}{g(k+1)} \tag{A.4}$$

which implies the upper bound since we already know from (A.2) that $\lim_{n\to\infty} \frac{f(k)}{g(k)} = 1$. Similarly the lower bound follows since $\frac{1}{2}g(1) \leq f(1)$ and Equation (A.4) tells us that the gap between $\frac{1}{2}g(k)$ and $f(k)$ grows with k. The straightforward calculation

$$\begin{aligned}
\frac{f(k)}{g(k)} \leq \frac{f(k+1)}{g(k+1)} &\iff \frac{f(k)}{f(k+1)} \leq \frac{g(k)}{g(k+1)} \\
&\iff \frac{(2k)!\left((k+1)!\right)^2}{(k!)^2(2k+2)!} \leq \frac{2^{2k}\sqrt{\pi(k+1)}}{\sqrt{\pi k}\,2^{2k+2}} \\
&\iff \frac{(k+1)^2}{(2k+2)(2k+1)} = \frac{k+1}{2(2k+1)} \leq \frac{\sqrt{k+1}}{4\sqrt{k}} \\
&\iff \frac{(k+1)^2}{4(2k+1)^2} \leq \frac{k+1}{16k} \\
&\iff 16k(k+1) \leq 4(2k+1)^2 = 16k^2 + 16k + 4.
\end{aligned}$$

concludes this argument.

A.4 Complexity Theory: Optimization vs. Approximation

Let us recall the following standard notions from basic complexity theory [30]: A *decision problem* Π is a subset of $\{0,1\}^*$. The elements of Π are supposed to be encodings of complex objects having a certain property in common. An algorithm ALG is said to *decide* Π if its output $\text{ALG}(x)$ on a $x \in \{0,1\}^*$ is "yes" if and only if $x \in \Pi$ and "no" otherwise.

An algorithm ALG is a *polynomial time algorithm* if its running time is bounded by a polynomial in the length of the input x. All decision problems Π admitting polynomial time algorithms constitute the class P. A decision problem Π is in class NP if there is a polynomial time algorithm ALG such that for any $x \in \Pi$ there is a *certificate* $y \in \{0,1\}^*$ with the properties

- The length of y is bounded by a polynomial in the length of x.
- $\text{ALG}(x,y) =$ "yes"

and for all $x \notin \Pi$ and $y \in \{0,1\}^*$ the output of ALG is "no". A decision problem Π is NP-hard if any problem $\Pi' \in$ NP can be reduced to Π in polynomial time.

However, in practical applications we often are concerned with optimization problems. Formally, an *optimization problem* consists of

1. A set of instances $\mathcal{I}$.
2. For each instance $I \in \mathcal{I}$ a set of *feasible solutions* S_I.
3. For each instance $I \in \mathcal{I}$ an *objective function* $c\colon S_I \to \mathbb{R}$, assigning each feasible solution a value (often interpreted as cost or profit).
4. An indicator whether the objective is to be minimized (cost interpretation) or maximized (profit interpretation).

We will only consider minimization problems. Furthermore, we adopt the convention that the name of an optimization problem denotes the optimal value for an instance. For example, $\textsc{ShortestPath}(G, u, v)$ is the length of a shortest path connecting u and v in G.

There is the following connection between decision problems and optimization problems: For an instance $I \in \mathcal{I}$ we can construct a canonical decision problem Π_I

$$\Pi_I := \{(s, k) \mid s \in S_I, c(s) \leq k\}$$

that formalizes the question "Is there a solution costing at most k?". We say that the optimization problem is NP-hard if Π_I is NP-hard. Since it is widely believed that $P \neq \text{NP}$, establishing the NP-hardness of an optimization problem is an indication that there possibly is no polynomial time algorithm and the optimization problem is therefore hard to solve.

Unfortunately, many interesting optimization problems are NP-hard. The usual way to circumvent this issue is to look for approximation algorithms: Instead of looking for optimal feasible solutions, one is comfortable with any feasible solution which cost is close to the optimal one. A polynomial time algorithm Alg is called *ϱ-approximation algorithm* for an optimization problem Opt if

$$\textsc{Alg}(I) \leq \varrho \cdot \textsc{Opt}(I)$$

for every instance I of Opt.

A.5 Results from Applied Probability

A.5.1 A Short Excursion to Queuing Theory

In Queuing Theory, one considers systems in which *requests* waiting for some kind of service arrive and are served by some server process. The arrival and service of

requests is modelled as a stochastic process. More precisely, requests arrive at times determined by a probability distribution. Similarly, request service times are also given by probability distributions and the assumption is made that each request has a service time according to this distribution.

The simplest case is when the time between successive arrivals of requests, called *interarrival time*, is given by an exponential distribution. Formally, $(X_n)_{n\in\mathbb{N}}$ is a family of random variables, all independently and identically distributed according to the exponential distributed with parameter λ. The so-called *arrival times* are then given by

$$\begin{aligned} T_0 &:= 0 \\ T_n &:= \sum\nolimits_{i=1}^{n} X_i \qquad n \in \mathbb{N}. \end{aligned}$$

The family $(N_t)_{t\in\mathbb{R}}$ of random variables counting the requests which arrived until time t (i. e., $N_t := \max\{n \in \mathbb{N}_0 \mid T_n \leq t\}$) is called *Poisson process with parameter* λ.

There is the following very important connection between a Poisson process with parameter λ and the Poisson distribution with parameter λ. A proof as well as an excellent introduction can be found in [27].

Theorem A.5 *Let t_1, t_2 be reals with $t_2 > t_1 \geq 0$. The number of requests N arriving between time t_1 and time t_2 is distributed as*

$$\text{Prob}\left[N = N_{t_2} - N_{t_1} = k\right] = \frac{\big(\lambda(t_2 - t_1)\big)^k}{k!} \mathrm{e}^{-\lambda(t_2 - t_1)}. \tag{A.5}$$

In other words: The distribution of N is the Poisson distribution with parameter $\lambda(t_2 - t_1)$. □

A.5.2 A Result on the Symmetric Random Walk

Another interesting concept intensively studied by the methods of Probability Theory is the so-called *symmetric random walk*: Suppose we have n independent random variables $X_1, \ldots, X_n$ with probability distribution

$$\text{Prob}\left[X_i = k\right] = \begin{cases} \frac{1}{2} & k = 1 \\ \frac{1}{2} & k = -1. \end{cases}$$

The sequence $W_j := \sum_{i=1}^{j} X_i$, $1 \leq j \leq n$, is called random walk because one can imagine it as moving n steps on a line, for each step deciding by a coin toss whether to move left or right.

There are many things known about the symmetric random walk, but we are mostly interested in the expected travel distance $\mathbb{E}[|W_n|]$. A detailed derivation can be found in [39] or at the web resource [40].

Theorem A.6 *Let W_n be a symmetric random walk of length n.*

$$\mathbb{E}[|W_n|] \sim \sqrt{\frac{2n}{\pi}}. \tag{A.6}$$

□

A.6 Tools from Analysis

The results mentioned in this section can be found in standard books on analysis, for example [36].

Theorem A.7 (Bernoulli's inequality) *For any real $x > -1$ and $n \in \mathbb{N}$ we have the relation*

$$(1+x)^n \geq 1 + nx.$$

□

A function $f\colon [a,b] \to \mathbb{R}$ is called *concave (on $[a,b]$)* if the condition

$$f(\theta a + (1-\theta)b) \geq \theta f(a) + (1-\theta)f(b)$$

holds for all $\theta \in [0,1]$. (Note that $\theta a + (1-\theta)b \in [a,b]$ for all such θ.)

Theorem A.8 (Jensen's inequality) *Let $f\colon [a,b] \to \mathbb{R}$ be a concave function. For arbitrary points $x_1, \ldots, x_n$ from $[a,b]$ and $\theta_i \in [0,1]$ such that $\sum_{i=1}^n \theta_i = 1$ the function f satisfies*

$$f\left(\sum\nolimits_{i=1}^n \theta_i x_i\right) \geq \sum\nolimits_{i=1}^n \theta_i f(x_i).$$

□

Bibliography

[1] Norbert Ascheuer, Martin Grötschel, Sven O. Krumke, and Jörg Rambau. Combinatorial online optimization. In *Proceedings of the International Conference of Operations Research Zürich (OR98)*, pages 21–37. Springer, 1999.

[2] Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. The online-transportation problem: Competitive scheduling of elevators. Technical report, Zuse Institute Berlin, 1998. Preprint SC 98-34.

[3] Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. Online Dial-a-Ride problems: Minimizing the completion time. In *Proceedings of the 17st Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 639–650. Springer, 2000.

[4] Mikhail J. Atallah and S. R. Kosaraju. Efficient solutions to some transportation problems with application to minimizing robot arm travel. *SIAM J. Comput.*, 17(5):819–869, 1988.

[5] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 184–193, 1996.

[6] Yair Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 161–168, 1998.

[7] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis for the multi-level feedback algorithm. *Math. Oper. Res.*, 31(1):85–108, 2006.

[8] Ralf Borndörfer, Martin Grötschel, Fridolin Klostermeier, and Christian Küttner. Telebus berlin: Vehicle scheduling in a Dial-a-Ride system. In Nigel Wilson, editor, *Proceedings of the 7th International Workshop on Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems, pages 391–422. Springer, 1999.

[9] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[10] Bernard Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM*, 47:1012–1027, 2002.

[11] John Clark and Derek A. Holton. *Graphentheorie – Grundlagen und Anwendungen*. Spektrum Lehrbuch. Spektrum Akademischer Verlag, 1994.

[12] Amin Coja-Oghlan, Sven O. Krumke, and Till Nierhoff. A heuristic for the stacker crane problem on trees which is almost surely exact. In *Proceedings of the 14th Annual International Symposium on Algorithms and Computation*, volume 2906 of *Lecture Notes in Computer Science*, pages 605–614. Springer, 2003.

[13] Amin Coja-Oghlan, Sven O. Krumke, and Till Nierhoff. Scheduling a server on a caterpillar network - a probabilistic analysis. In *Proceedings of the 6th Workshop on Models and Algorithms for Planning and Scheduling Problems*, 2003.

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2nd edition, 2001.

[15] Willem E. de Paepe. Complexity results and competitive analysis for vehicle routing problems. Technische Universiteit Eindhoven, 2002. Ph. D. Thesis.

[16] Willem E. de Paepe, Jan K. Lenstra, Jiri Sgall, René A. Sitters, and Leen Stougie. Computer-aided complexity classification of Dial-a-Ride problems. *INFORMS Journal on Computing*, 16(2):120–132, 2004.

[17] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 448–455, 2003.

[18] Greg N. Frederickson and D. J. Guan. Nonpreemptive ensemble motion planning on a tree. *J. Algorithms*, 15(1):29–60, 1993.

[19] Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. *SIAM J. Comput.*, 7(2):178–193, 1978.

[20] Hiroshi Fujiwara and Kazuo Iwama. Average-case competitive analyses for ski-rental problems. In *ISAAC 2002*, volume 2518 of *Lecture Notes in Computer Science*, pages 476–488. Springer, 2002.

[21] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics – A Foundation for Computer Science*. Addison Wesley, 2nd edition, 1994.

[22] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal of Computation*, 13(2):338–355, 1984.

[23] Dietrich Hauptmeier. Online algorithms for transport systems. Technische Universität Berlin, 1999. Diplom Thesis.

[24] Dietrich Hauptmeier, Sven O. Krumke, and Jörg Rambau. The online Dial-a-Ride problem under reasonable load. In *CIAC 2000*, volume 1767 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2000.

[25] Svante Janson, Tomasz Łuczak, and Andrzej Ruciński. *Random Graphs*. Interscience Series in Discrete Mathematics and Optimization. Wiley, 2000.

[26] Philip Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *Journal of Algorithms*, 19(1):104–115, July 1995.

[27] Leonard Kleinrock. *Queuing Systems*, volume 1: Theory. John Wiley & Sons, 1975.

[28] Sven O. Krumke. Online optimization – competitive analysis and beyond. Technische Universität Berlin, 2001. Habilitation Thesis.

[29] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[30] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[31] Seth Pettie. Finding minimum spanning trees in $\mathcal{O}(m\alpha(m,n))$ time. Technical Report CS-TR-99-23, University of Texas, Austin, 1999.

[32] Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM*, 49:16–34, 2002.

[33] Mark Scharbrodt, Thomas Schickinger, and Angelika Steger. A new average case analysis for completion time scheduling. In *Proceedings of the 34th ACM Symposium on Theory of Computing,* Montréal, QB, pages 170–178, 2002.

[34] B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM Journal of Computation*, 17(6):1253–1262, 1988.

[35] A. Souza-Offtermatt and Angelika Steger. The expected competitive ratio for weighted completion time scheduling. In *Proceedings of the 21st Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Computer Science*, pages 620–631, 2004.

[36] Uwe Storch and Hartmut Wiebe. *Lehrbuch der Mathematik, Band 1*. Spektrum Lehrbuch. Spektrum Akademischer Verlag, 3rd edition, 2003.

[37] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[38] Silvia Vogel. Stochastik für Informatiker. Lecture notes of summer term 2001, TU Ilmenau.

[39] Eric W. Weisstein. *CRC Concise Encyclopedia of Mathematics*. CRC Press, 1998.

[40] Eric W. Weisstein. Random walk – 1-dimensional. From MathWorld – A Wolfram Web Resource, 2005. `http://mathworld.wolfram.com/RandomWalk1-Dimensional.html`.

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit der probabilistischen Analyse eines elementaren Spezialfalls von sogenannten *Dial-a-Ride-Problemen*. Dabei handelt es sich um allgemeine Transportprobleme, die z. B. das Traveling-Salesman-Problem umfassen.

Allgemein haben Dial-a-Ride-Probleme folgende Struktur: Eine Flotte von Fahrzeugen muß eine Reihe von Transportaufträgen bedienen, wobei jeder Transportauftrag aus einem Gut besteht, das von einem Ursprungsort zu einem Zielort zu transportieren ist. Dazu können die Fahrzeuge ein Transportnetzwerk, z. B. ein Straßensystem, benutzen. Wesentlich ist, daß durch die Wahl von Wegen im Transportnetzwerk Kosten entstehen, die minimiert werden sollen. Hinzu kommen häufig Nebenbedingungen, wie z. B. beschränkte Kapazität und / oder Reichweite der Fahrzeuge, Bedingungen an die Reihenfolge der Aufträge und so weiter.

Wir betrachten ein spezielles Dial-a-Ride-Problem, das bei der Optimierung von industriellen Aufzugsystemen aufgetreten ist. Es hat die folgenden Charakteristiken:

- Es gibt nur ein Fahrzeug, das höchstens einen Auftrag gleichzeitig bearbeiten kann.
- Ein Auftrag wird bedient, indem das Fahrzeug zum Ursprungsort fährt, das Gut auflädt und es zum Zielort transportiert. Ein einmal aufgeladenes Gut muß zum Zielort gebracht werden (nicht-präemptive Transporte).
- Es gibt ein Depot, von dem aus das Fahrzeug seine Fahrt beginnt und zu dem es am Ende zurückkehren muß.
- Das zugrundeliegende Transportnetzwerk ist ein (graphentheoretischer) Baum.
- Ziel ist es, die Gesamtbedienungszeit zu minimieren.

und wird kurz mit DARP bezeichnet. Es ist bekannt, daß DARP sogar für sehr einfache Bäume NP-schwer ist [23].

Zunächst geben wir ein graphentheoretisches Modell für DARP an. Anschließend beschreiben wir einen effizienten Approximationsalgorithmus von Frederickson und

Guan [18]. Herz dieses Algorithmus ist die sogenannte *Balancierungstechnik*, die aus einzelnen Aufträgen größere (Teil-)Touren von Aufträgen erzeugt, die jede Gesamttour enthalten muß. Diese Balancierungstechnik ist auch von grundlegender Bedeutung für die späteren probabilistischen Analysen. Um zu einer Gesamttour zu gelangen, müssen die Teiltouren mit minimalen Kosten verbunden werden, was durch Lösen eines ebenfalls NP-schweren Steinerbaum-Problems erreicht werden kann. Daher wird dieser Schritt mit Hilfe der *Minimaler-Spannbaum-Heuristik* gelöst, was auf den $\frac{4}{3}$-Approximationsalgorithmus USE-MST führt. Ferner zeigen wir, daß USE-MST eine optimale Lösung liefert, wenn die Teiltouren eine *Sternmetrik* bilden.

Nach einer kurzen Wiederholung der wahrscheinlichkeitstheoretischen Grundlagen widmen wir uns der probabilistischen Analyse von USE-MST. Dazu betrachten wir ausführlich das Ergebnis von Coja-Oghlan, Krumke und Nierhoff [12], wonach die Teiltouren mit hoher Wahrscheinlichkeit eine Sternmetrik bilden, wobei ein bestimmtes allgemeines Wahrscheinlichkeitsmodell vorausgesetzt wird. Daraus folgt sofort, daß USE-MST auf fast allen Eingaben optimale Lösungen liefert. Wir konnten ein Lemma verbessern, das die Anzahl der Teiltouren abschätzt.

Dial-a-Ride-Probleme sind in der Praxis häufig durch eine weitere Schwierigkeit gekennzeichnet, den sogenannten *Online-Aspekt*: Die Aufträge werden erst nach und nach bekannt, müssen jedoch möglichst bald bearbeitet werden. Dieses Nicht-Wissen über die Zukunft kann momentan günstige Lösungen in der weiteren Entwicklung sehr schlecht werden lassen, weshalb nach guten *Online-Algorithmen* gesucht wird.

Das Standardwerkzeug zur Analyse von Online-Algorithmen ist die *kompetitive Analyse*, die die Ergebnisse von Online-Algorithmen mit denen optimaler *Offline-Algorithmen* vergleicht, die „in die Zukunft schauen“ können, die also alle Aufträge im Voraus kennen. Dies ist aber oft eine viel zu starke Annahme, weswegen mehrere Varianten der kompetitiven Analyse entwickelt wurden.

Unser Ansatz besteht in einer Kombination von probabilistischer Analyse und kompetitiver Analyse, also *probabilistischer kompetitiver Analyse*: Wir möchten wissen, wie gut ein Online-Algorithmus bei fast allen Eingaben gegenüber dem Offline-Algorithmus sein kann, wenn die Eingaben zufällig erzeugt werden. Wir untersuchen in diesem Zusammenhang die IGNORE-*Strategie* [2] unter Verwendung von USE-MST genauer und können ein erstes probabilistisches Kompetivitätsresultat zeigen. Außerdem zeigen wir, daß sich IGNORE gutartig gegenüber zufällig erzeugten Eingaben verhält, wohingegen die REPLAN-Strategie im allgemeinen deren zufällige Struktur verfälscht.

Thesen

1. Die Eindeutigkeit von Wegen in Bäumen gestattet eine einfache und nützliche Beschreibung der Lösungsstruktur des DARP auf Bäumen: Eine Lösung besteht aus Auftragsbögen, balancierenden Bögen und verbindenden Bögen.

2. Eine Menge balancierender Bögen kann in linearer Zeit berechnet werden. Zusammen mit der MST-Heuristik führt das auf einen effizienten $\frac{4}{3}$-Approximationsalgorithmus USE-MST.

3. Der Algorithmus USE-MST erzeugt optimale Lösungen (empirisch).

4. Für ein geeignetes Wahrscheinlichkeitsmodell der Eingaben kann gezeigt werden: USE-MST erzeugt asymptotisch fast sicher optimale Lösungen, ðwenn viele Aufträge vorhanden sind, werden fast alle Eingaben optimal gelöst.

5. Grundlage für dieses Ergebnis ist ein modifiziertes Wahrscheinlichkeitsmodell, das es erlaubt, die „Verbindungsinformation“ und die „Balancierungsinformation“ der Aufträge getrennt zu analysieren. Damit kann gezeigt werden, daß die Balancierungsoperation die Aufträge zu wenigen Komponenten „zusammenklebt“, die dann eine Stern-Metrik bilden.

6. Man kann die klassische kompetitive Analyse auf zufällige Eingaben ausdehnen und damit probabilistische Kompetitivität definieren.

7. Wenn die Aufträge gemäß der Wahrscheinlichkeitsverteilung erzeugt werden, die der Analyse von USE-MST zugrunde liegt, und ferner Ankunftszeiten in beliebiger Weise erzeugt werden, so haben die von der IGNORE-Strategie erzeugten Probleme die gleiche Wahrscheinlichkeitsstruktur. Dies gilt im Allgemeinen nicht für die REPLAN-Strategie.

8. Für Auftragssequenzen, die durch mit Parameter λ exponentialverteilte Zwischenankunftszeiten erzeugt werden, erzeugt IGNORE(USE-MST) Lösungen, die

für große Auftragszahlen nicht teurer sind als optimale Offline-Lösungen. Vorausgesetzt wird, daß die Aufträge schneller ankommen als sie abgearbeitet werden.

9. Probabilistische kompetitive Analyse ist für die Bewertung von Online-Algorithmen für den praktischen Einsatz geeigneter als klassische worst-case kompetitive Analyse.

Benjamin Hiller
Ilmenau, den 05. Juli 2004

Hiermit erkläre ich an Eides statt, diese Diplomarbeit selbständig und eigenhändig sowie nur unter Zuhilfenahme der angegebenen Quellen angefertigt zu haben.

Benjamin Hiller
Ilmenau, den 05. Juli 2004