Karlsruhe Institute of Technology

On Preprocessing the ALT-Algorithm

Student Thesis of

Fabian Fuchs

At the faculty of Computer Science
Institute for Theoretical Informatics (ITI)

Reviewer:	Prof. Dr. Dorothea Wagner
Advisor:	Dipl.-Math. Reinhard Bauer
Second advisor:	Dr. Giacomo Nannicini

22. März 2010 – 12. Juli 2010

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderung entnommen wurde.

Karlsruhe, den 12. Juli 2010

Ort, Datum (Fabian Fuchs)

Abstract

In this thesis, we study the preprocessing phase of the ALT algorithm. ALT is a well known preprocessing-based speed-up technique for Dijkstra's algorithm, which allows fast computations of shortest paths in large scale road networks. The preprocessing of the ALT algorithm has some degree of freedom, in that it must select a subset of nodes in the graph, called landmarks, that fulfill a special role; optimally choosing these landmarks is NP-hard, hence no effective exact solution algorithm exists. In this thesis, we study the landmark selection process; we propose several solution methods, including a greedy algorithm as well as a new heuristic. Furthermore, we propose a new model for the search space of an ALT shortest path computation, which allows us to reduce the problem of optimally choosing k landmarks to the maximum coverage problem, for which approximation results are known, and to formulate the landmark selection problem as an integer linear program.

Abstract

In dieser Studienarbeit wurde die Vorberechnungsphase des ALT Algorithmus' untersucht. ALT ist ein bekannter und in der Literatur mehrfach untersuchter Routenplanungsalgorithmus der auf Dijkstra's Algorithmus basiert aber durch einen Vorberechnungsschritt kürzeste Wege Anfragen wesentlich schneller beantworten kann. Im Vorbereitungsschritt verbleiben gewisse Freiheitsgrade in der Wahl von speziellen Punkten, sogenannten Landmarken, zu und von denen die Entfernung zu allen anderen Knoten vorberechnet wird. Es ist NP hart diesen Freiheitsgrad optimal zu füllen, das heißt eine optimale Menge an Landmarken auszuwählen, weshalb dafür bisher Heuristiken verwendet werden. In dieser Arbeit haben wir die Wahl der Landmarken untersucht und neue Lösungsalgorithmen vorgestellt, darunter ein Greedy Algorithmus sowie eine neue Heuristik. Des weiteren schlagen wir ein neues Suchraummodel vor, das es uns ermöglicht das Problem auf ein 'Maximales Abdeckungsproblem' (maximum coverage problem) zu reduzieren, für das Approximationsergebnisse existieren und das es uns ermöglicht das Problem der optimalen Wahl der Landmarken als Ganzzahliges Lineares Programm zu schreiben.

Contents

1. Introduction

The computation of shortest paths on graphs is a problem with many real-life applications like route planning in an internet or car navigation system, traffic simulation or logistic optimization. One prominent example is, that one is interested in the shortest path between two given endpoints: one departure node s and a destination node t. This variant of the problem, which is called the *single-source single-target* problem, can be solved in polynomial time with the well known Dijkstra's algorithm (Dijkstra, 1959), assuming that the graph has non-negative edge weights. If this condition does not hold but the graph does not contain negative cycles, then the shortest path problem can be solved with the Bellman-Ford algorithm (Bellman, 1958; Ford and Fulkerson, 1962). Finding a simple shortest path in graphs with negative cycles is shown to be NP-hard. Indeed, these two algorithms can be used to compute not only the shortest path between two given endpoints, but all shortest paths originating from a source node s: this variant of the problem is called *single-source*. Although this is a very commonly used variant, there exist other variants of the shortest path problem, such as the *all-pairs* shortest paths problem, which requires the computation of the shortest path between all pairs of nodes in a given graph. The all-pairs shortest paths problem can be solved with the Floyd-Warshall algorithm (Cormen et al., 2001).

One interesting practical application of the single-source single-target shortest path problem is route planning: indeed, road networks can be easily represented as graphs, and the problem of finding the shortest route between two points can be solved by computing the shortest path between two nodes on the corresponding graph. In many cases, one would be interested in computing shortest paths in a matter of few milliseconds: one typical example is a server scenario which has to deal with several user shortest paths queries per second, and is therefore required to carry out each computation in a very short time. However, the graphs which represent real-world road networks, such as the European or the North American road networks, can have very large sizes, and the classical shortest paths algorithms introduced above may require several seconds for each computation. For the practical applications that we have in mind, this is not acceptable. To deal with this issue, we can employ *speed-up techniques* that preprocess the input data, in order to accelerate the answer to single-source single-target shortest paths queries (Wagner and Willhalm, 2007). This means that the solution method has two phases: a preprocessing phase, that computes

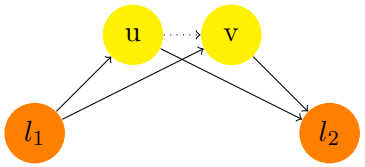

Figure 1.1: triangle inequality

useful information on the input graph and is applied *only once*, and a query phase, which computes the actual shortest paths using the output of the preprocessing phase to accelerate the search. There are many different preprocessing based variants of Dijkstra's algorithm, such as ALT (Goldberg and Harrelson, 2004), Arc-Flags (Gutman, 2004), Contraction Hierarchies (Geisberger et al., 2008), Highway Node Routing (Holzer et al., 2009; Schultes and Sanders, 2007), SHARC (Bauer and Delling, 2009) and Reach Based Pruning (Goldberg et al., 2007).

Research in this field is, at least for enhanced problems such as time-dependent graphs, still very active, with the objective of improving preprocessing time and space, as well as the performance of the query phase. An overview of several speed-up techniques and some experimental work can be found in (Delling et al., 2009; Wagner and Willhalm, 2007).

In this thesis we will take a closer look at the preprocessing phase required by the ALT algorithm, which has been introduced by (Goldberg and Harrelson, 2004). ALT stands for A* search, Landmarks and Triangle inequality, as these are the main ingredients of the algorithm. The A* algorithm is a generalization of Dijkstra's algorithm, which uses a function (the *potential* function π) to estimate distances between nodes in the graph. A good potential function π can be used to reduce the search space (i.e. the set of nodes that have to be "explored" before the solution is found) of the shortest path queries effectively. One way to define a potential function is through the use of *landmarks* (Goldberg and Harrelson, 2004), that is, a subset of the nodes in the graph which fulfill a special role. For these landmarks, the distances to/from all other nodes in the graph are computed during the preprocessing phase, and estimations of distances within the graph can then be carried out by means of the triangle inequality. The triangle inequality states that, for any three nodes u, v, l in the graph, $d(u, v) \leq d(u, l) + d(l, v)$, where $d(u, v)$ is the distance between u and v, i.e. the length of the shortest path between those two nodes. This inequality can be used to derive bounds: as illustrated in Figure 1.1, the distance from u to v is smaller than $\text{dist}(l_1, v) - \text{dist}(l_1, u)$ as well as $\text{dist}(u, l_2) - \text{dist}(v, l_2)$. One question arises: how do we select landmarks on the input graph?

As it is shown in (Bauer et al., 2010), most of the preprocessing based speed-up techniques for Dijkstra's algorithm (such as ALT, Arc-Flags, SHARC, Highway Node Routing and Contraction Hierarchies), have some degrees of freedom which are NP-hard to determine optimally, and are therefore heuristically determined in practice. This also applies for the ALT algorithm, for which it is shown that selecting a set of k landmarks which minimizes the expected search space for random single-source single-target queries is NP-hard. Several heuristics for this purpose have been proposed in the literature, such as: *random*, *farthest* and *avoid* introduced by Goldberg and Harrelson (Goldberg and Harrelson, 2004), *advanced avoid* introduced by Delling et al (Delling et al., 2006), as well as *maxCover* introduced by Goldberg and Werneck (Goldberg and Werneck, 2005).

In this thesis we will study the problem of selecting the set of landmarks in a graph so that the search space of random single-source single-target shortest path queries is minimized. We will focus on simple graph classes such as paths and trees, and give results on the effect that landmarks have on the shortest paths computations. We will also give an upper bound on the number of landmarks which are needed to achieve minimal search spaces in trees. We will then analyze the performance of a greedy algorithms for the problem of selecting k landmarks, and introduce a new heuristic. We propose an integer linear program (ILP) formulation that models the problem of selecting the best possible set of landmarks on a given graph. Finally, we carry out computational experiments to compare the performance of the methods studied in this thesis with existing algorithms from the literature.

2. Preliminaries

We denote by

$\mathbb{N}$	the set of non-negative integers
$\mathbb{R}^+$	the set of positive real numbers, including zero
$\lvert A\rvert$	the cardinality of the set A

A *graph* is an ordered pair $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ comprising a set $\mathbf{V}$ of *vertices* and a set $\mathbf{E} \subset \mathbf{V} \times \mathbf{V}$ of *edges*. We also consider a non-negative *length function* $\text{len} : \mathbf{E} \to \mathbb{R}^+$; for the sake of simplicity, we write $\text{len}(u, v) := \text{len}((u, v))$.

Definition and Remark 2.1. For a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

1. $\mathbf{G}$ is said to be *directed* if the elements of $\mathbf{E}$ are ordered pairs, and is called *undirected* if such pairs are unordered.
2. For a directed graph, we say that an edge $e = (u, v)$ *leaves* node u and *enters* node v.
3. Within this thesis, an undirected graph is equivalent to a directed graph such that for every edge $e = (u, v) \in \mathbf{E}$ there exists an edge $e' = (v, u) \in \mathbf{E}$.
4. For an undirected graph, the length function is required to be symmetric: $\text{len}(u, v) = \text{len}(v, u)$.

Definition 2.2. Given a graph $G = (\mathbf{V}, \mathbf{E})$ and two nodes $u, v \in \mathbf{V}$

1. A *path from u to v* (also called a *u-v-path*) is a sequence $(v_0, v_1, \ldots, v_p)$ of vertices such that $u = v_0$,$v = v_p$, and there exists an edge $(v_{i-1}, v_i) \in \mathbf{E}$ for every $i \in \{1, \ldots, p\}$.
2. A path from u to v is called *simple* if no vertices are repeated on the path.
3. The *distance* between two vertices u and v is defined as

$$\text{dist}(u, v) := \min_{(v_0,\ldots,v_p) \text{ is a path from u to v}} \sum_{i=1}^{p} \text{len}(v_{i-1}, v_i).$$

   and if no path from u to v exists in $\mathbf{G}$, $\text{dist}(u, v) := \infty$ by convention.
4. Given $u, s, t \in \mathbf{V}$, and an s-t-path Q, we define the distance between u and Q as $\text{dist}(u, Q) := \min_{v \in Q}\{\text{dist}(s, v)\}$.

5. A *cycle* is a path $(v_0, v_1, \ldots, v_r)$ with $v_0 = v_r$.
6. If we require the path $(v_0, v_1, \ldots, v_{r-1})$ to be simple we say $(v_0, v_1, \ldots, v_r)$ is a *simple cycle*.

Definition and Remark 2.3. For a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

1. We say $\mathbf{G}$ is connected if for each pair $(u, v) \in \mathbf{V} \times \mathbf{V}$ a path from u to v exists.
2. $\mathbf{G}$ is a *tree* if $\mathbf{G}$ is connected and without cycles.
3. In a tree any two vertices are connected by exactly one shortest path.
4. we call $\mathbf{G}$ a *chain* if for $\mathbf{V} = \{v_0, \ldots, v_n\}$ the set of edges is $\mathbf{E} = \{(v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n)\}$, intuitively speaking the graph is one path.

Dijkstra's Algorithm

Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with non-negative length function $\text{len}(\cdot, \cdot)$ and a source s and a target t with $s, t \in \mathbf{V}$, Dijkstra's algorithm computes the distances and shortest paths from the source to the target.

During the run of the algorithm a vertex can have three different states: It is either unreached, reached or settled. It is unreached while the distance label is set to infinity, reached as soon as a tentative shortest path from the source to the vertex has been found and settled once it has been processed by the algorithm (Step 1 and 2). Note that a shortest path from s to a vertex v has been found if v is settled. If v is reached, a tentative path from s to v has been found which is not necessarily shortest.

The algorithm initializes a distance label $\text{dist}(v) = \infty$ for every vertex $v \in \mathbf{V} \backslash \{s\}$ and $\text{dist}(s)$ to 0. We use a min-based priority queue Q to manage the reached vertices priorized by the distance label. Then, Step 1 and Step 2 are executed until the priority queue Q is empty which is equally to that every reachable vertex has been settled.

Step 1. Select the reached vertex v with smallest distance label $\text{dist}(v)$. If there is more than one with the same distance label there is some freedom left in how to break ties. The most common way is to compare the vertex's ID and select the vertex with the smallest ID.
Step 2. For every edge from v to a vertex u, update the distance label $\text{dist}(u)$ if $\text{dist}(v) + \text{len}(v, u)$ is smaller than the current distance label $\text{dist}(u)$ to $\text{dist}(u) = \text{dist}(v) + \text{len}(v, u)$. If the distance label is updated, the algorithm found a new tentative shortest path from s to the vertex u. Hence we update $\text{parent}(u)$: $\text{parent}(u) = v$ and add u to priority queue Q if it has not yet been added, i.e. if u has been unreached so far. Finally the state of the vertex v is changed to settled.

For every vertex v such that its state is settled, the distance from s to v is $\text{dist}(s, v) = \text{dist}(v)$. If a vertex's distance label is still set to infinity after the algorithm ended this vertex can not be reached from s. The shortest path from s to a settled vertex v is encoded in the $\text{parent}(\cdot)$ label, which must be followed from v to s to decode the shortest path from s to v.

Definition 2.4. In a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with positive length function $\text{len}(\cdot, \cdot)$ we call the set of vertices which are settled during the run of Dijkstra's algorithm for an s-t-query the *Dijkstra search space* of this s-t-query and denote it by $\mathbf{V}_{Dij}(s, t)$.

A* search

Algorithm 2.1 DIJKSTRA(G, s, t)

Require: graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and Vertices $s, t\mathbf{V}$
Ensure: $\text{dist}(s, v) = \text{dist}(v) \forall v \in \mathbf{V}$
1: **for all** $v \in \mathbf{V}$ **do**
2: $\quad \text{parent}(v) \leftarrow \perp$
3: $\quad \text{state}(v) \leftarrow$ unreached
4: $\quad \text{dist}(v) \leftarrow \infty$
5: $\text{dist}(s) \leftarrow 0$
6: $\text{state}(s) \leftarrow$ reached
7: **while** vertex v with $\text{state}(v) =$ reached exists and $\text{state}(t) \neq$ reached **do**
8: $\quad$ Select $v \in \mathbf{V}$ with $\text{state}(v) =$ reached and minimal $\text{dist}(v)$
9: $\quad$ **for all** $u \in \mathbf{V}$ with $(v, u) \in \mathbf{E}$ **do**
10: $\quad\quad$ **if** $\text{dist}(v) + \text{len}(v, u) < \text{dist}(u)$ **then**
11: $\quad\quad\quad \text{dist}(u) \leftarrow \text{dist}(v) + \text{len}(v, u)$
12: $\quad\quad\quad \text{parent}(u) \leftarrow v$
13: $\quad\quad\quad \text{state}(u) \leftarrow$ reached
14: $\quad \text{state}(v) \leftarrow$ settled

For a given graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with non-negative length function $\text{len}(\cdot, \cdot)$, a source $s \in \mathbf{V}$, a target $t \in \mathbf{V}$ and a so called *potential* function $\pi_t : \mathbf{V} \to \mathbb{R}^+$, *A*-search* computes the distance from s to t. A* is a generalization of Dijkstra's algorithm, which uses a potential function π_t in addition to the distance function to select the node which should be visited next. The potential function $\pi_t(v)$ is usually a lower bound for the distance from any vertex v to the target t.

The potential function π_t has to be *feasible*: $\text{dist}(u, v) - \pi_t(u) + \pi_t(v) \geq 0$ for all vertices $u, v \in \mathbf{V}$. This implies that the resulting new edge "lengths" $\overline{\text{len}}(u, v) := \text{len}(u, v) - \pi_t(u) + \pi_t(v)$ are still positive, which is necessary for Dijkstra's algorithm as well as A* search to work correctly. With these new edge lengths Dijkstra is equal to A* search, which is proved in section 3.1.

A* works exactly as Dijkstra, except that the algorithm stops once the target is settled and that in *Step 1* the reached vertex which has minimal costs $\widehat{\text{cost}}(v) := \text{dist}(v) + \pi_t(v)$ is selected. In general, we write $\text{cost}(v) := \text{dist}(s, v) + \pi_t(v)$ which is equal to $\widehat{\text{cost}}(v)$ for all settled nodes v after the execution of A* search with source s.

ALT algorithm

The ALT algorithm is a variant of A* search, whose main idea is to use landmarks and triangle inequality to compute a feasible potential function. Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with non-negative length function $\text{len}(\cdot, \cdot)$, ALT first selects a set $\mathbf{L} \subset \mathbf{V}$ of k landmarks and precomputes the distance to and from these landmarks for every vertex in $\mathbf{V}$ during a preprocessing step.

With the triangle inequality $|x+y| \leq |x|+|y|$ we can derive the two inequalities $\text{dist}(u, v) \geq \text{dist}(l, v) - \text{dist}(l, u)$ and $\text{dist}(u, v) \geq \text{dist}(u, l) - \text{dist}(v, l)$ for any $u, v, l \in \mathbf{V}$. Note that the right hand sides only involve distances to and from a landmark which we assumed to be available thanks to the preprocessing. With the convention $\infty - \infty := 0$, we can therefore define

$$\pi_t^{l+}(v) := \text{dist}(v, l) - \text{dist}(t, l) \tag{2.1}$$

$$\pi_t^{l-}(v) := \text{dist}(l, t) - \text{dist}(l, v) \tag{2.2}$$

Algorithm 2.2 A*(G, s, t, π_t)

Require: graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, Vertices s and t
Ensure: parent(t) encodes the shortest path from s to t

```
1: for all v ∈ V do
2:    parent(v) ← ⊥
3:    state(v) ← unreached
4:    dist(v) ← ∞
5: dist(s) ← 0
6: state(s) ← reached
7: while vertex v with state(v) = reached exists and state(t) ≠ reached do
8:    Select v ∈ V with state(v) = reached and minimal cost(v) = dist(v) + π_t(v)
9:    for all u ∈ V with (v, u) ∈ E do
10:      if dist(v) + len(v, u) + π_t(u) < dist(u) + π_t(u) then
11:         parent(u) ← v
12:         dist(u) ← dist(v) + len(v, u)
13:         state(u) ← reached
14:   state(v) ← settled
```

as feasible potential functions. To get the tightest bounds we use the maximum of these potential functions for all landmarks and define

$$\pi_t^{\mathbf{L}}(v) := \max_{l \in \mathbf{L}} \{\pi_t^{l+}(v), \pi_t^{l-}(v)\} \tag{2.3}$$

as the potential function ALT uses for the A* search algorithm.

Definition 2.5. In a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with positive length function $\text{len}(\cdot, \cdot)$ we call the set of vertices which are settled during the run of ALT for an s-t-query the *ALT search space* of this s-t-query. Since the search space of an s-t-query of ALT depends on the set of landmarks, we denote the search space in the ALT algorithm by $\mathbf{V_L}(s, t)$.

Note that we did not specify how to break ties when extracting nodes from the prioriry queue Q, i.e. the case where multiple nodes attain the minimum on line 8 of Algorithm 2.3. This can have an impact on the search space: see Section 3.1.

Algorithm 2.3 ALT(G, s, t)

Require: graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, Vertices s and t
Ensure: parent(t) encodes the shortest path from s to t

1: $\mathbf{L}$ = generate_Landmarks($\mathbf{G}$, k) {select set of k landmark}
2: **for all** $v \in \mathbf{V}$ **do**
3: parent$(v) \leftarrow \perp$
4: state$(v) \leftarrow$ unreached
5: dist$(s, v) \leftarrow \infty$
6: dist$(s, s) \leftarrow 0$
7: state$(s) \leftarrow$ reached
8: **while** vertex v with state$(v) =$ reached exists and state$(t) \neq$ reached **do**
9: Select $v \in \mathbf{V}$ with state$(v) =$ reached and minimal cost$(v) =$ dist$(s, v) + \pi_t^{\mathbf{L}}(v)$
10: **for all** $u \in \mathbf{V}$ with $(v, u) \in \mathbf{E}$ **do**
11: **if** dist$(s, v) +$ len$(v, u) + \pi_t^{\mathbf{L}}(u) <$ dist$(s, u) + \pi_t^{\mathbf{L}}(u)$ **then**
12: parent$(u) \leftarrow v$
13: dist$(s, u) \leftarrow$ dist$(s, v) +$ len(v, u)
14: state$(u) \leftarrow$ reached
15: state$(v) \leftarrow$ settled

3. Landmarks

The selection of good landmarks is a crucial point for the performance of the ALT algorithm. Since good landmarks yield a tight lower bound on the distance from any vertex to the target, with better landmarks usually fewer vertices must be settled by the ALT algorithm. Therefore, the search space of these s-t-queries can be reduced.

In order to compare different sets of landmarks, we need a measure of quality. We decided to use the sum of the sizes of the search spaces over all s-t-queries as a measure of quality for a given set of landmarks $\mathbf{L}$:

$$\sum_{s,t \in \mathbf{V}} V_{\mathbf{L}}(s,t) \tag{3.1}$$

Minimizing this quantity is equivalent to minimizing the average size of the search space for a random s-t-query. This results in the following minimization problem. Solving this problem and therefore minimizing the average size of the search space for a random s-t-query will be our objective in this thesis.

Definition 3.1 (MinALT)**.** Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and an integer k, the problem MinALT is the problem of selecting k landmarks so that the resulting overall ALT search space is minimal, i.e.to select a set $\mathbf{L} \subset \mathbf{V}$ so that $\sum_{s,t \in \mathbf{V}} \mathbf{V}_{\mathbf{L}}(s,t)$ is minimal.

3.1 Search space model

In order to measure the search space, we need a model we can compute without having to run the algorithm itself. Therefore, in literature, a search space model was first described by (Bauer et al., 2010). It is motivated by the proof of Theorem 4.1 in (Goldberg and Harrelson, 2004).

Theorem (Goldberg 4.1 [incorrect])**.** *Let π_t and π'_t be two feasible potential function such that $\pi_t(t) = \pi'_t(t) = 0$ and for any vertex $\pi'_t(v) \geq \pi_t(v)$ (i.e. π'_t dominates π_t). Then the set of vertices scanned by A* search search using π'_t is a subset of the set of vertices scanned by A* search search using π_t.*

This is not true, since for example let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a undirected graph with $\mathbf{V} = \{0,1,2,3,4,5\}$ and $\mathbf{E} = \{(0,2),(0,3),(1,3),(1,4),(1,5),(2,4),(2,5),(3,4),(3,5)\}$ with edge lengths 1 except for the edges $(2,4),(2,5)$ and $(3,5)$ which have edge lengths 2. For simplicity every undirected edge is given only once. The graph is illustrated in Figure 3.1.

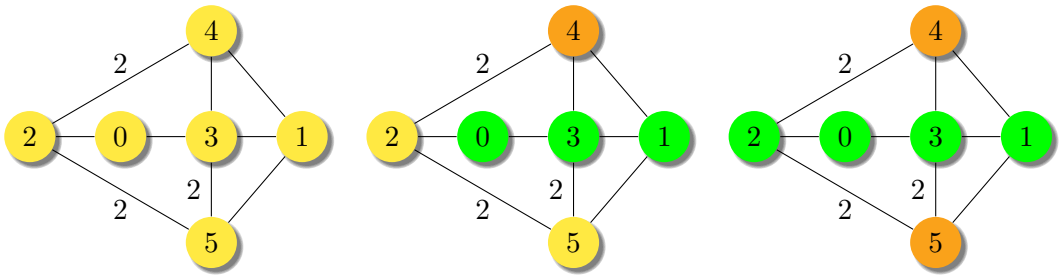

Figure 3.1: A counterexample to Theorem 4.1 in (Goldberg and Harrelson, 2004). On the left the graph in which the counterexample can be constructed is shown. All edge lengths are 1 if not stated otherwise. In the graph in the middle the ALT search space of the 0-1-query using the landmark 4 is illustrated. On the right the same search space is shown using the landmarks $\{4, 5\}$.

For a query from source vertex $s = 0$ to sink vertex $t = 1$, we will first examine the search space for the potential $\pi_t^{\mathbf{L}}$ induced by the landmark $\mathbf{L} = \{4\}$.

In the first step of the ALT algorithm, the to the vertex 0 adjacent vertices 2 and 3 are reached. $\hat{\text{cost}}(2) = \text{dist}(2)+\pi_t^{\mathbf{L}}(2) = 1+1 = 2$ whereas $\hat{\text{cost}}(3) = \text{dist}(3)+\pi_t^{\mathbf{L}}(3) = 1+0 = 1$, hence 3 is settled next. Then, the target 1 is reached with $\hat{\text{cost}}(1) = \text{dist}(1) + \pi_t^{\mathbf{L}}(1) = 2+0 = 2$. There is a tie between vertices 1 and 2, but vertex 1 is settled since $1 < 2$. The search space is therefore $\{0, 3, 1\}$.

Using the potential $\pi_t^{\mathbf{L}'}$ induced by the landmarks $\mathbf{L}' = \{4, 5\}$ in the first step of the ALT algorithm the costs for vertex 2 are the same but for vertex 3 they change to $\hat{\text{cost}}(3) = \text{dist}(0, 3) + \pi_t^{\mathbf{L}'}(3) = 1 + 1 = 2$. Since the costs are the same the tie is broken by settling the one with the smaller ID. Hence 2 is settled first, then 3 will get settled and afterwards the target vertex 1. The resulting search space is $\{0, 2, 3, 1\}$.

One can easily check that the potential $\pi_t^{\mathbf{L}'}$ dominates $\pi_t^{\mathbf{L}}$ since the maximum over the potentials induced by the landmarks is taken as described in Chapter 2.

The search space deduced from the way Goldberg breaks ties, would be

$$\begin{aligned}\widetilde{\mathbf{V}}_{\mathbf{L}}(s,t) = \{v \in \mathbf{V} |\, &\text{dist}(s,v) + \pi_t^{\mathbf{L}}(v) < \text{dist}(s,t) \text{ or} \\ &\text{dist}(s,v) + \pi_t^{\mathbf{L}}(v) = \text{dist}(s,t) \text{ and } v \leq t\}\end{aligned} \tag{3.2}$$

for a s-t-query in a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with a set of landmarks $\mathbf{L}$.

In contrast to $\widetilde{\mathbf{V}}_{\mathbf{L}}(s,t)$, the search space model we will work with is the worst case search space model, which can be expressed as

$$\overline{\mathbf{V}}_{\mathbf{L}}(s,t) = \{v \in \mathbf{V} |\, \text{dist}(s,v) + \pi_t^{\mathbf{L}}(v) \leq \text{dist}(s,t)\} \tag{3.3}$$

If it is not clear which graph $\mathbf{G}$ is meant, we will write $\overline{\mathbf{V}}_{\mathbf{L},\mathbf{G}}(s,t)$.

In the following theorem we show, that using the worst case search space model is reasonable.

Theorem 1. *Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a graph, $\mathbf{L}$ a set of landmarks and the length function be strictly positive:* $\text{len}(u, v) > 0$ *for any* $u, v \in \mathbf{V}$.
Assuming ALT breaks ties based on the labeling of vertices, then there is a labeling of $\mathbf{G}$ such that the ALT search space $\mathbf{V}_{\mathbf{L}}(s,t)$ equals $\overline{\mathbf{V}}_{\mathbf{L}}(s,t)$.

To show the theorem we first show that executing the ALT algorithm is equivalent to executing Dijkstra's algorithm with altered length function. Then, we show that the theorem holds for Dijkstra with altered length function.

Lemma. *On a graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *with non-negative length function* $\text{len}(\cdot,\cdot)$*, executing A* search with the feasible potential* π *is equivalent to Dijkstra's algorithm with altered length function* $\overline{\text{len}}(u,v) := \text{len}(u,v) + \pi(v) - \pi(u)$*.*
This means that the nodes are settled in the same order for both algorithms. We assume that ties are broken in the same way for both algorithms.

Proof. During the initialization both algorithms do the same (state(v)=unreached, $\text{dist}(v) = \infty$ / $\overline{dist}(v) = \infty$, for $v \in \mathbf{V}$ and $\text{dist}(s) = \overline{dist}(s) = 0$).
For both algorithms the loop executes as long as there are unsettled vertices and the target t is not settled.
In the loop Dijkstra selects the vertex v with smallest distance label $\overline{\text{dist}}(v)$ whereas on the other hand A* search selects the vertex v with smallest $\text{dist}(v) + \pi_t(v)$.
We will prove by induction that both algorithms settle the nodes in the same order which means that $\overline{\text{dist}}(v)$ and $\text{dist}(v) + \pi_t(v)$ are the same up to a constant factor c: $\overline{\text{dist}}(v) = \text{dist}(v) + \pi_t(v) - \pi_t(s)$ with $c = -\pi_t(s)$.

Base case: The first settled vertex is in both algorithms s and $\overline{\text{dist}}(s) = 0 = \text{dist}(s) + \pi_t(s) - \pi_t(s)$.

Let the first r vertices be settled in the same order by both algorithms and $\overline{\text{dist}}(v) = \text{dist}(v) + \pi_t(v) - \pi_t(s)$ for all vertices v.
Induction step: Let v be the $(r+1)$th vertex which is settled by Dijkstra's algorithm. Then $\overline{\text{dist}}(v) \overset{induction}{=} \text{dist}(v) + \pi_t(v) - \pi_t(s)$ must be minimal and hence $\text{dist}(v) + \pi_t(v) = \overline{\text{dist}}(v) + \pi_t(s)$ is minimal. In case there is a tie in both algorithms the same vertex is chosen, too. Then the distances are updated for any to v adjacent vertex u if $\text{dist}(v) + \text{len}(v,u) < \text{dist}(u)$ in both algorithms. A* search updates the distance to $\text{dist}(u) = \text{dist}(v) + \text{len}(u,v)$ and Dijkstra to $\overline{\text{dist}}(u) = \overline{\text{dist}}(v) + \overline{\text{len}}(v,u) = \text{dist}(v) + \pi_t(v) - \pi_t(s) + \text{len}(v,u) + \pi_t(u) - \pi_t(v) = \text{dist}(v) + \text{len}(v,u) + \pi_t(u) - \pi_t(s) = \text{dist}(u) + \pi_t(u) - \pi_t(s)$. And therefore both algorithms are settling the same $r+1$th vertex and $\overline{\text{dist}}(v) = \text{dist}(v) + \pi_t(v) - \pi_t(s)$ after settling the $r+1$th vertex.

Since both algorithms end once the target is settled the lemma is shown. □

Proof of Theorem 1. Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with $\mathbf{L}, \pi_t^{\mathbf{L}}$ as in the theorem, $\overline{\text{len}}(u,v) := \text{len}(u,v) + \pi_t^{\mathbf{L}}(v) - \pi_t^{\mathbf{L}}(u)$ the length function and vertices $s, t \in \mathbf{V}$ and an node labeling $\alpha : \mathbf{V} \to \mathbb{N}$ such that

$$\alpha(v) \neq \alpha(u) \text{ if } v \neq u$$
$$\alpha(t) > \alpha(v) \text{ for all } v \in \mathbf{V}$$

We have to show that $\mathbf{V}_{Dij}^{\mathbf{G}^\alpha}(s,t) = \overline{\mathbf{V}}_{\mathbf{L},\mathbf{G}}(s,t)$.

"$\subseteq$": Let therefore be $s, t, v \in \mathbf{V}$ such that $v \in \mathbf{V}_{Dij}^{\mathbf{G}^\alpha}(s,t)$, then:

case $\text{dist}(s,v) \leq \text{dist}(s,t)$**:** $v \in \overline{\mathbf{V}}_{\mathbf{L},\mathbf{G}}(s,t)$ due to the definition of $\overline{\mathbf{V}}_{\mathbf{L},\mathbf{G}}(s,t)$.

case $\text{dist}(s,v) > \text{dist}(s,t)$**:** t is settled before v is settled and therefore $v \notin \mathbf{V}_{Dij}^{\mathbf{G}^\alpha}(s,t)$, i.e. this case is not possible.

"$\supseteq$": Let $s, t, v \in \mathbf{V}$ such that $v \in \overline{\mathbf{V}}_{\mathbf{L},\mathbf{G}}(s,t)$, then:

case $\text{dist}(s,v) + \pi_t^{\mathbf{L}}(v) < \text{dist}(s,t)$**:** Dijkstra settles all vertices with $\text{dist}(s,v) + \pi_t^{\mathbf{L}}(v) - \pi_t^{\mathbf{L}}(s) < \text{dist}(s,t) - \pi_t^{\mathbf{L}}(s)$ before t is settled $\Rightarrow v \in \mathbf{V}_{Dij}^{\mathbf{G}^\alpha}(s,t)$.

case $\text{dist}(s,v) + \pi_t^{\mathbf{L}}(v) = \text{dist}(s,t)$**:** For a path Q from s to v it holds for all vertices $w \in Q$ that $\text{dist}(s,w) + \pi_t^{\mathbf{L}}(w) \leq \text{dist}(s,t)$. Hence it is sufficient to show that a path Q from s to v with vertices in the Dijkstra search space exists such that $t \notin Q$.
We first show that $t \notin Q$: Assume t is on the path Q then $\text{dist}(s,v) + \pi_t^{\mathbf{L}}(v) = \text{dist}(s,t) + \text{dist}(t,v) + \pi_t^{\mathbf{L}}(v) \overset{\text{dist}(t,v)>0}{>} \text{dist}(s,t)$ and hence v is not in $\overline{\mathbf{V}}_{\mathbf{L},\mathbf{G}}(s,t)$.
Now we show that the (shortest) path from s to v (including v) is in the Dijkstra search space. We prove that by induction over the vertices $s = w_0, w_1, \ldots, w_r = v$ on the path Q.
Base case: $s = w_0$ is in the Dijkstra search space.
Induction step: If $\text{dist}(s,w_i) + \pi_t^{\mathbf{L}}(w_i) < \text{dist}(s,t)$ the first case applies and hence $w_i \in \mathbf{V}_{Dij}^{\mathbf{G}^\alpha}(s,t)$.
If $\text{dist}(s,w_i) + \pi_t^{\mathbf{L}}(w_i) = \text{dist}(s,t)$ all vertices w_j with $j < i$ are in the Dijkstra search space and hence the distance label of the vertex w_i is $\text{dist}(w_i) = \text{dist}(s,w_i)$. Since $\text{dist}(s,w_i) + \pi_t^{\mathbf{L}}(w_i) <= \text{dist}(s,t)$ there is a tie and Dijkstra settles w_i since $\alpha(w_i) < \alpha(t)$.

□

Remark 3.2. From the theorem follows, that the search space of the ALT algorithm for an s-t-query can not generally (i.e. if the vertex labeling is unknown) be bound tighter than $\{v \in \mathbf{V} | \, \text{dist}(s,v) + \pi_t^{\mathbf{L}}(v) \leq \text{dist}(s,t)\}$ using a total order function $<$ comparing a vertex's ID to the target's ID to break ties on a graph with strictly positive length function.

Now that we justified our new search space model $\overline{\mathbf{V}}_{\mathbf{L}}(s,t)$, we show that the statement of theorem 4.1 from (Goldberg and Harrelson, 2004) is correct for our search space model.

Definition 3.3. On a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with positive length function, a set of landmarks $\mathbf{L}$ and feasible potential $\pi_t^{\mathbf{L}}$ we write $\overline{\mathbf{V}}_{\pi_t^{\mathbf{L}}}(s,t)$ if it is not clear which potential is meant.

Theorem 2. *Let π_t and π'_t be two feasible potential function such that $\pi_t(t) = \pi'_t(t) = 0$ and for any vertex $\pi'_t(v) \geq \pi_t(v)$ (i.e. π'_t dominates π_t). Then the set of vertices in the search space $\overline{\mathbf{V}}_{\pi'_t}(s,t)$ settled by ALT using π'_t is a subset of the set of vertices in the search space $\overline{\mathbf{V}}_{\pi_t}(s,t)$ settled by ALT using π_t.*

Proof. Suppose that a vertex v is not settled by ALT using the potential $\overline{\mathbf{V}}_{\pi_t}(s,t)$. It is sufficient to show that v is not settled by ALT using the potential $\overline{\mathbf{V}}_{\pi'_t}(s,t)$. Since v is not settled, it must be that $\text{dist}(s,v) + \pi_t(v) > \text{dist}(s,t)$. Since π'_t dominates π_t it holds that $\text{dist}(s,v) + \pi'_t(v) \geq \text{dist}(s,v) + \pi_t(v) > \text{dist}(s,t)$ and hence v is not settled by ALT using the potential $\pi'_t(v)$. □

3.2 Effect of landmarks on chains

In this section, we show the effect landmarks have on ALT queries on chain graphs.

Definition 3.4. The search space of a graph is *optimal* if the overall search space $\sum_{s,t \in \mathbf{V}} \overline{\mathbf{V}}_{\mathbf{L}}(s,t)$ is minimal, i.e. it can not be improved any further.

Theorem 3. *Using only one landmark, a graph $\mathbf{G}$ consisting of only one chain of vertices $(v_0, v_1, \ldots, v_n)$, $n \geq 3$ with strictly positive length function* len*, has optimal search space, if and only if v_0 or v_n is a landmark*

Proof. "→": Let $\mathbf{G}$ be a graph as in Theorem 3.

Assume that the landmark is $\mathbf{L} = \{v_j\}$, $j \neq 0$, $j \neq n$. Consider the query with source v_j and the target be the to v_j adjacent vertex to which the connecting edge has a higher length. W.l.o.g this be v_{j+1}. Then the costs of v_{j-1} and v_{j+1} are

$$\begin{aligned}\text{cost}(v_{j-1}) &= \text{dist}(v_j, v_{j-1}) + \pi^{\mathbf{L}}_{v_j+1}(v_{j-1}) \\ &= \text{dist}(v_j, v_{j-1}) + \max\{\text{dist}(v_{j-1}, v_j) - \text{dist}(v_{j+1}, v_j), \text{dist}(v_{j+1}, v_j) - \text{dist}(v_{j-1}, v_j)\} \\ &= \text{dist}(v_j, v_{j-1}) + \text{dist}(v_{j+1}, v_j) - \text{dist}(v_{j-1}, v_j) \\ &= \text{dist}(v_{j+1}, v_j)\end{aligned}$$

and

$$\begin{aligned}\text{cost}(v_{j+1}) &= \text{dist}(v_j, v_{j+1}) + \pi^{\mathbf{L}}_{v_j+1}(v_{j+1}) \\ &= \text{dist}(v_j, v_{j+1}) + \max\{\text{dist}(v_{j+1}, v_j) - \text{dist}(v_{j+1}, v_j), \text{dist}(v_{j+1}, v_j) - \text{dist}(v_{j+1}, v_j)\} \\ &= \text{dist}(v_j, v_{j+1})\end{aligned}$$

and therefore both are in $\overline{\mathbf{V}}_{\mathbf{L}}(s,t)$ and the query is not optimal.

"$\leftarrow$":

Let $\mathbf{G}$ be such a graph and w.l.o.g. v_0 be the landmark (otherwise relabel the chain). We show that all s-t-queries have a minimal search space.

Let therefore $s, t \in \mathbf{V}$ be two vertices such that the search space of the s-t query is not minimal without using landmarks, i.e. $\mathbf{V}_{Dij}(s,t) \neq (s = u_0, u_1, \ldots, u_r = t)$. It is shown in $\rightarrow$ that such a query exists (e.g. from the source v_1 to its adjacent vertex with higher costs).

We know that without using landmarks at least the one vertex adjacent to the source which is not on the s-t-path must be settled (otherwise the query would be optimal, since behind the target no vertex is in the search space due to the strictly positive length function). Let the source be $s = v_i$ and the target $t = v_l$ then either v_{i+1} or v_{i-1} is in the Dijkstra search space but not on the s-t-path. In the first case $l < i$ and in the latter $l > i$. In the first case the costs for v_{i+1} are

$$\begin{aligned}\text{cost}(v_{i+1}) &= \text{dist}(v_i, v_{i+1}) + \pi_t(v_{i+1}) \\ &= \text{dist}(v_i, v_{i+1}) + \max\{\text{dist}(v_{i+1}, v_0) - \text{dist}(t, v_0), \text{dist}(v_0, t) - \text{dist}(v_0, v_{i+1})\} \\ &= \text{dist}(v_i, v_{i+1}) + \text{dist}(v_{i+1}, v_0) - \text{dist}(t, v_0) \\ &= \text{dist}(v_i, v_{i+1}) + \text{dist}(v_{i+1}, t) \\ &= \text{dist}(v_i, t) + 2\,\text{dist}(v_i, v_{i+1}) \\ &= \text{dist}(s, t) + 2\,\text{dist}(v_i, v_{i+1})\end{aligned}$$

and in the latter case the costs for v_{i-1} are

$$\begin{aligned}\text{cost}(v_{i-1}) &= \text{dist}(v_i, v_{i-1}) + \pi_t(v_{i-1}) \\ &= \text{dist}(v_i, v_{i-1}) + \max\{\text{dist}(v_{i-1}, v_0) - \text{dist}(t, v_0), \text{dist}(v_0, t) - \text{dist}(v_0, v_{i-1})\} \\ &= \text{dist}(v_i, v_{i-1}) + \text{dist}(v_0, t) - \text{dist}(v_0, v_{i-1})\} \\ &= \text{dist}(v_i, v_{i-1}) + \text{dist}(v_{i-1}, t) \\ &= \text{dist}(v_i, t) + 2\,\text{dist}(v_i, v_{i-1}) \\ &= \text{dist}(s, t) + 2\,\text{dist}(v_i, v_{i-1})\end{aligned}$$

and hence in both cases the vertex is not in the search space since the costs are higher then those of t.

$$\begin{aligned}\text{cost}(t) &= \text{dist}(s, t) + \pi_t(t) \\ &= \text{dist}(s, t) + \max\{\text{dist}(t, v_0) - \text{dist}(t, v_0), \text{dist}(v_0, t) - \text{dist}(v_0, t)\} \\ &= \text{dist}(s, t)\end{aligned}$$

□

3.3 Effect of landmarks on trees

In this section we discuss the effect of landmarks on trees. We first describe the effect the potentials $\pi_t^{l+}(v)$ and $\pi_t^{l-}(v)$ of one landmark l have on shortest path search spaces. Afterwards we show, which s-t-queries have optimal search spaces in trees and finally give some insight of how vertices on paths between landmarks can be seen.

3.3.1 Potential $\pi_t^{l+}(v)$ in undirected trees

For undirected trees, we show in this subsection that the search space includes only vertices with distance less than $\text{dist}(t, s\text{-}l\text{-path})$ from the *core* of an s-t-query away for the landmark $l \in \mathbf{L}$ with minimal distance $\text{dist}(t, s\text{-}l\text{-path})$.

Definition 3.5. The *core* of an s-t-query with respect to a landmark l is the intersection of the s-t-path and the s-l-path for a landmark $l \in \mathbf{L}$. We write $\mathcal{C}_{s,t}^l := \{v \in \mathbf{V} | v \in s\text{-}t\text{-path and } v \in s\text{-}l\text{-path}\}$ for the core of an s-t-query with landmark l.

Note that the core of an s-t-query with landmark l is a path itself and $\text{dist}(v, \mathcal{C}_{s,t}^l) = \min\{\text{dist}(v,u) | u \in \mathcal{C}_{s,t}^l\}$ is the distance from v to the closest vertex in the core.

Theorem 4. *Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a undirected graph and $s, t \in \mathbf{V}$ and l be the landmark with minimal distance $\text{dist}(t, \mathcal{C}_{s,t}^l)$. Using only the potential π_t^{l+} the search space of an s-t-query includes all vertices in the core of the s-t-query with respect to l and all vertices v such that $\text{dist}(v, \mathcal{C}_{s,t}^l) \leq \text{dist}(t, \mathcal{C}_{s,t}^l)$, i.e. their distance to the core $\mathcal{C}_{s,t}^l$ is less or equal to the distance from t to the core $\mathcal{C}_{s,t}^l$.*

Proof. Let $s, t \in \mathbf{V}$ and l be the landmark with minimal distance $\text{dist}(t, \mathcal{C}_{s,t}^l)$. It holds that

$$\overline{\mathbf{V}}_{\mathbf{L}}(s,t) = \{v \in \mathbf{V} | \, \text{dist}(s,v) + \pi_t^{l+} \leq \text{dist}(s,t)\} \tag{3.4}$$

Hence we have to show that

$$\text{dist}(s,v) + \pi_t^{l+} \leq \text{dist}(s,t) \Leftrightarrow \text{dist}(v, \mathcal{C}_{s,t}^l) \leq \text{dist}(t, \mathcal{C}_{s,t}^l) \tag{3.5}$$

to prove that $\overline{\mathbf{V}}_{\mathbf{L}}(s,t) = \{v \in \mathbf{V} | \, \text{dist}(v, \mathcal{C}_{s,t}^l) \leq \text{dist}(t, \mathcal{C}_{s,t}^l)\}$.

"⇒": Let $v \in \overline{\mathbf{V}}_{\mathbf{L}}(s,t)$ then it holds that

$$\begin{aligned}
&\text{dist}(s,v) + \pi_t^{l+}(v) && \leq \text{dist}(s,t) \\
\overset{\pi_t^{l+} \geq 0}{\Rightarrow} \; &\text{dist}(s,v) + \text{dist}(v,l) - \text{dist}(t,l) && \leq \text{dist}(s,t) \\
\overset{(*)}{\Rightarrow} \; &\text{dist}(s,v) + \text{dist}(v,l) - \text{dist}(t, \mathcal{C}_{s,t}^l) - \text{dist}(\mathcal{C}_{s,t}^l, l) && \leq \text{dist}(s,t) \\
\Rightarrow \; &\text{dist}(s,v) + \text{dist}(v,l) - \text{dist}(\mathcal{C}_{s,t}^l, l) - \text{dist}(s,t) && \leq \text{dist}(t, \mathcal{C}_{s,t}^l)
\end{aligned}$$

where (*) follows since on the path from t to l is only one vertex c with $c \in \mathcal{C}_{s,t}^l$. Let Q denote the s-l path then we can distinguish the following two cases:

case $\text{dist}(v, \mathcal{C}_{s,t}^l) = \text{dist}(v, Q)$**:** Then let $\hat{c}$ be the last vertex in the intersection of the s-v-path and the s-l-path Q. In this case $\hat{c}$ is in the core $\mathcal{C}_{s,t}^l$ and it holds that

$$\text{dist}(s,v) + \text{dist}(v,l) = \text{dist}(s,l) + 2\,\text{dist}(v, \mathcal{C}_{s,t}^l) \tag{3.6}$$

We can follow

$$\begin{aligned} &\operatorname{dist}(s,v)+\operatorname{dist}(v,l)-\operatorname{dist}(\mathcal{C}^l_{s,t},l)-\operatorname{dist}(s,t) && \leq \operatorname{dist}(t,\mathcal{C}^l_{s,t}) \\ \overset{\text{eq 3.6}}{\Rightarrow} &\operatorname{dist}(s,l)+2\operatorname{dist}(v,\mathcal{C}^l_{s,t})-\operatorname{dist}(\mathcal{C}^l_{s,t},l)-\operatorname{dist}(s,t) && \leq \operatorname{dist}(t,\mathcal{C}^l_{s,t}) \end{aligned}$$

and by separating $\operatorname{dist}(v,\mathcal{C}^l_{s,t})$ on the left side

$$\begin{aligned} &\operatorname{dist}(v,\mathcal{C}^l_{s,t}) && \leq \frac{\operatorname{dist}(t,\mathcal{C}^l_{s,t})+\operatorname{dist}(\mathcal{C}^l_{s,t},l)+\operatorname{dist}(s,t)-\operatorname{dist}(s,l)}{2} \\ \Rightarrow &\operatorname{dist}(v,\mathcal{C}^l_{s,t}) && \leq \frac{\operatorname{dist}(t,c)+\operatorname{dist}(c,l)+\operatorname{dist}(s,c)+\operatorname{dist}(c,t)-\operatorname{dist}(s,c)-\operatorname{dist}(c,l)}{2} \\ \Rightarrow &\operatorname{dist}(v,\mathcal{C}^l_{s,t}) && \leq \frac{2\operatorname{dist}(t,c)}{2} \\ \Rightarrow &\operatorname{dist}(v,\mathcal{C}^l_{s,t}) && \leq \operatorname{dist}(t,\mathcal{C}^l_{s,t})) \end{aligned}$$

$\Rightarrow v \in \{v \in \mathbf{V} |\, \operatorname{dist}(\mathcal{C}^l_{s,t},v) \leq \operatorname{dist}(\mathcal{C}^l_{s,t},t)\}$.

case $\operatorname{dist}(v,\mathcal{C}^l_{s,t}) > \operatorname{dist}(v,Q)$**:** Let $\hat{c}$ again be the last vertex in the intersection of the $s-v$-path and the s-l-path Q and $\{s=v_0,v_1,\ldots,v_i=c,\ldots,v_r=t\}$ be the path Q from s to l. Let $v_j=\hat{c}$ then $j>i$ for this case since otherwise $\operatorname{dist}(v,\mathcal{C}^l_{s,t})=\operatorname{dist}(v,Q)$. Hence c must be on the path from s to v and it must hold that

$$\begin{aligned} &\operatorname{dist}(s,v) && \leq \operatorname{dist}(s,t) \\ \Rightarrow &\operatorname{dist}(s,c)+\operatorname{dist}(c,v) && \leq \operatorname{dist}(s,c)+\operatorname{dist}(c,t) \\ \Rightarrow &\operatorname{dist}(c,v) && \leq \operatorname{dist}(c,t) \\ \Rightarrow &\operatorname{dist}(\mathcal{C}^l_{s,t},v) && \leq \operatorname{dist}(\mathcal{C}^l_{s,t},t) \end{aligned}$$

$\Rightarrow v \in \{v \in \mathbf{V} |\, \operatorname{dist}(\mathcal{C}^l_{s,t},v) \leq \operatorname{dist}(\mathcal{C}^l_{s,t},t)\}$.

"$\Leftarrow$": Let v be in $\{v \in \mathbf{V} |\, \operatorname{dist}(\mathcal{C}^l_{s,t},v) \leq \operatorname{dist}(\mathcal{C}^l_{s,t},t)\}$ and let additionally Q denote the s-l path then obviously

$$\operatorname{dist}(\mathcal{C}^l_{s,t},v) \leq \operatorname{dist}(\mathcal{C}^l_{s,t},t) \tag{3.7}$$

and we can estimate that

$$\operatorname{dist}(s,v)+\operatorname{dist}(v,l)=\operatorname{dist}(s,l)+2\operatorname{dist}(v,Q) \leq \operatorname{dist}(s,l)+2\operatorname{dist}(v,\mathcal{C}^l_{s,t}) \tag{3.8}$$

By starting with the a true fact, we follow that $\operatorname{dist}(s,v)+\pi^{l+}_t(v) \leq \operatorname{dist}(s,t)$

$$\begin{aligned} &\operatorname{dist}(t,\mathcal{C}^l_{s,t}) && \geq \operatorname{dist}(t,\mathcal{C}^l_{s,t}) \\ \Rightarrow &\operatorname{dist}(t,\mathcal{C}^l_{s,t})+\operatorname{dist}(l,\mathcal{C}^l_{s,t})-\operatorname{dist}(l,\mathcal{C}^l_{s,t})+\operatorname{dist}(s,c)-\operatorname{dist}(s,c) && \geq \operatorname{dist}(t,\mathcal{C}^l_{s,t}) \\ \Rightarrow &\operatorname{dist}(l,\mathcal{C}^l_{s,t})+\operatorname{dist}(s,c)+\operatorname{dist}(t,\mathcal{C}^l_{s,t})-\operatorname{dist}(s,c)-\operatorname{dist}(l,\mathcal{C}^l_{s,t}) && \geq \operatorname{dist}(t,\mathcal{C}^l_{s,t}) \\ \Rightarrow &\operatorname{dist}(l,\mathcal{C}^l_{s,t})+\operatorname{dist}(s,t)-\operatorname{dist}(s,l) && \geq \operatorname{dist}(t,\mathcal{C}^l_{s,t}) \\ \Rightarrow &\operatorname{dist}(s,l)-\operatorname{dist}(l,\mathcal{C}^l_{s,t})-\operatorname{dist}(s,t) && \leq -\operatorname{dist}(t,\mathcal{C}^l_{s,t}) \\ \overset{\text{eq 3.8}}{\Rightarrow} &\operatorname{dist}(s,v)+\operatorname{dist}(v,l)-2\operatorname{dist}(v,\mathcal{C}^l_{s,t})-\operatorname{dist}(l,\mathcal{C}^l_{s,t})-\operatorname{dist}(s,t) && \leq -\operatorname{dist}(t,\mathcal{C}^l_{s,t}) \\ \Rightarrow &\operatorname{dist}(s,v)+\operatorname{dist}(v,l)-\operatorname{dist}(l,\mathcal{C}^l_{s,t})-\operatorname{dist}(s,t) && \leq 2\operatorname{dist}(v,\mathcal{C}^l_{s,t})-\operatorname{dist}(t,\mathcal{C}^l_{s,t}) \\ \overset{\text{eq 3.7}}{\Rightarrow} &\operatorname{dist}(s,v)+\operatorname{dist}(v,l)-\operatorname{dist}(l,\mathcal{C}^l_{s,t})-\operatorname{dist}(s,t) && \leq \operatorname{dist}(t,\mathcal{C}^l_{s,t}) \\ \Rightarrow &\operatorname{dist}(s,v)+\operatorname{dist}(v,l)-\operatorname{dist}(l,\mathcal{C}^l_{s,t})-\operatorname{dist}(t,\mathcal{C}^l_{s,t}) && \leq \operatorname{dist}(s,t) \\ \Rightarrow &\operatorname{dist}(s,v)+\operatorname{dist}(v,l)-\operatorname{dist}(l,t) && \leq \operatorname{dist}(s,t) \end{aligned}$$

If $\text{dist}(v,l) - \text{dist}(l,t) \geq 0$ we can easily follow that $\text{dist}(s,v) + \text{dist}(v,l) - \text{dist}(l,t) = \text{dist}(s,v) + \pi_t^{l+}(v) \leq \text{dist}(s,t)$ and showed that $v \in \overline{\mathbf{V}}_{\mathbf{L}}(s,t)$. Otherwise $\text{dist}(v,l) - \text{dist}(l,t) < 0$ and therefore $\text{dist}(s,v) > \text{dist}(s,t)$. We show that this can not be by distinguishing the following two cases:

case $c \in s$-v-**path:** $\text{dist}(s,c) + \text{dist}(c,v) = \text{dist}(s,v) > \text{dist}(s,t) = \text{dist}(s,c) + \text{dist}(c,t) \Rightarrow \text{dist}(c,v) > \text{dist}(c,t) \Rightarrow \text{dist}(\mathcal{C}_{s,t}^l, v) > \text{dist}(\mathcal{C}_{s,t}^l, t)$. Contradiction since $\text{dist}(\mathcal{C}_{s,t}^l, v) \leq \text{dist}(\mathcal{C}_{s,t}^l, t)$.

case $c \notin s$-v-**path:** $\underbrace{\text{dist}(s,\hat{c})}_{\leq \text{dist}(s,c)} + \underbrace{\text{dist}(\hat{c},v)}_{\leq \text{dist}(c,t)} = \text{dist}(s,v) > \text{dist}(s,t) = \text{dist}(s,c) + \text{dist}(c,t)$ with $\hat{c}$ being the last vertex on the intersection of the s-v-path and the s-l-path. Contradiction since $\text{dist}(s,\hat{c}) + \text{dist}(\hat{c},v) \leq \text{dist}(s,t)$.

And hence $v \in \overline{\mathbf{V}}_{\mathbf{L}}(s,t)$.

□

3.3.2 Potential $\pi_t^{l-}(v)$ in undirected trees

In this section we examine the influence of $\pi_t^{l-}(v) = \text{dist}(l,t) - \text{dist}(l,v)$ on undirected trees. We know, that vertices v for which $\text{dist}(s,v) + \pi(v) > \text{dist}(s,t)$ are not settled. We show now, that when the potential $\pi_t^{l-}(v)$ is used vertices which are closer to the landmark than necessary are not settled, i.e. a vertex v which is on a s-l-path for $l \in \mathbf{L}$ but not in the core $\mathcal{C}_{s,t}^l$ of the s-t-query is not settled.

Theorem 5. *Given a tree $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with strictly positive length function and a set $\mathbf{L}$ of landmarks. In an s-t-query no vertex closer to the landmark than necessary are in the search space $\overline{\mathbf{V}}_{\mathbf{L}}(s,t)$, i.e. if a vertex v is on a path from s to a landmark l and not on the path from s to t it is not settled.*

Proof. Let c be the vertex connecting the s-t-path with the landmark l, then $\text{cost}(c) = \text{dist}(s,c) + \pi_t^{l-}(c) = \text{dist}(s,c) + \text{dist}(l,t) - \text{dist}(l,c) = \text{dist}(s,c) + \text{dist}(c,t) = \text{dist}(s,t) = \text{cost}(t)$.

Let now v be a vertex on a path from s to a landmark l and not on the path from s to t, then $\text{cost}(v) = \text{dist}(s,v) + \pi_t^{l-}(v) = \text{dist}(s,v) + \text{dist}(l,t) - \text{dist}(l,v) = \text{dist}(s,c) + \text{dist}(c,v) + \text{dist}(l,c) + \text{dist}(c,t) - \text{dist}(l,v) = \text{dist}(s,t) + \text{dist}(l,c) + \text{dist}(c,v) - \text{dist}(l,v) > \text{dist}(s,t) = \text{cost}(t)$. □

Hence a vertex which is closer to the landmark than necessary will not be settled.

3.3.3 Undirected trees with combined potential $\pi_t^{\mathbf{L}}()$

In Section 3.3.1 we showed for undirected trees that vertices v with $\text{dist}(v, \mathcal{C}_{s,t}^l) > \text{dist}(t, \mathcal{C}_{s,t}^l)$ (with $l \in \mathbf{L}$ so that $\text{dist}(t, \mathcal{C}_{s,t}^l)$ is minimal) are not settled for the s-t-query using the potential π_t^{l+}. In Section 3.3.2 we showed for undirected trees that vertices v on the path from s to a landmark l but not on the path from s to t are not settled.

By combining both potentials π_t^{l+} and π_t^{l-} for all landmarks $l \in \mathbf{L}$ we obtain the potential $\pi_t^{\mathbf{L}}$ which is used by the ALT algorithm. The maximum over the potentials is taken, both effects are combined and we can infer that the ALT algorithm is not settling vertices v such that they are more than $\text{dist}(t, \mathcal{C}_{s,t}^l)$ away from the core $\mathcal{C}_{s,t}^l$ of an s-t-query (with $l \in \mathbf{L}$ such that $\text{dist}(t, \mathcal{C}_{s,t}^l)$ is minimal) as well as it is not settling vertices which are closer to a landmark than necessary.

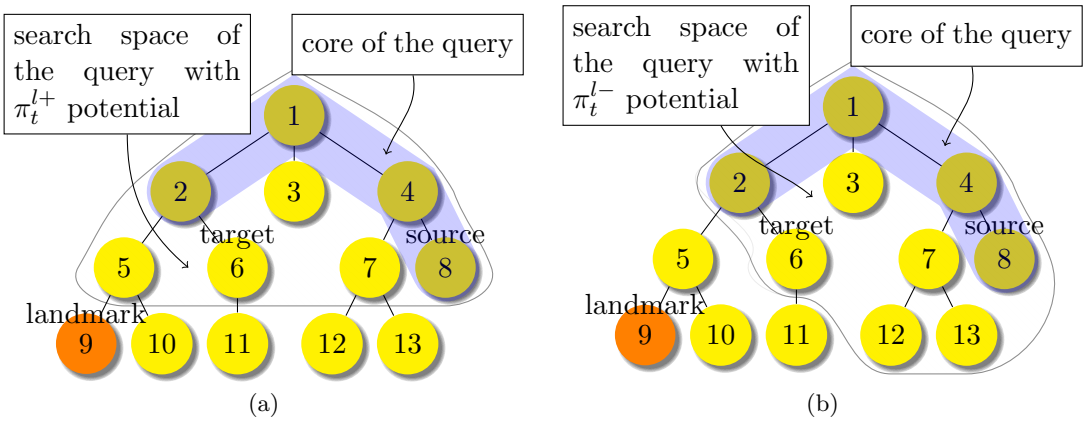

Figure 3.2: Illustration of the search space of the query from 8 to 6 with the landmark 9. On the left using only the potential π_t^{l+} and on the right using only the potential π_t^{l-}.

3.3.4 Optimal search spaces in trees

Definition 3.6. For an s-t-query on a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ we say that this query has an *optimal search space* if the search space of the ALT algorithm is equivalent to the path from s to t.

In this section we give all cases so that the search space of an s-t-query is optimal and we show, that the ALT query is optimal only in these cases. We can derive an algorithm to compute an upper bound on how many landmarks are needed to have optimal search spaces from the following Theorem 6.

For Theorem 6 we limit the tree to uniform edge lengths of length 1 for simplicity. For general edge lengths the cases 2 and 3 need to be modified and the distance from the target to the core of an s-t-query must be considered.

Theorem 6. *On a tree with uniform edge lengths 1 and the landmark l as root, an s-t-query is optimal if and only if:*

1. *t is one of the vertices on the path from s to the landmark l.*
2. *The* connecting vertex *c has a maximum degree of 3, s has degree 1 and all other vertices on the s-t-path have maximum degree 2, except for t which may have any degree.*
3. *The path from s to t has no vertex with degree > 2, except for t which may have any degree.*

Where the connecting vertex *c is the first vertex which is in the intersection of the path from t to l and the path from s to l.*

Proof. " $\rightarrow$ " : Let q be the path from s to the landmark l.

1. t is one of the vertices on the path from s to the landmark l.

   Since $\text{dist}(t, q) = 0$ follows with Theorem 4 that the search space is optimal.

2. The *connecting vertex* c has a maximum degree of 3, s has degree 1 and all other vertices on the s-t-path have maximum degree 2, except for t which may have any degree.

The source and all vertices except for the connecting vertex c must not be considered, since ALT can not reach vertices which are not on the s-t-path from them. From the fact that the connecting vertex has degree 3, we know that the one adjacent vertex which is not on the s-t-path is on the path from the connecting vertex to the landmark. That this vertex will not be settled follows from Theorem 5 which states that ALT will not search closer to the landmark then necessary

3. The path from s to t has no vertex with degree > 2, except for t which may have any degree.

   In this case ALT has only one possibility to settle vertices not on the s-t-path: To settle the adjacent vertex u of s which is not on q. Either this vertex u is on the path from s to the landmark and therefore will not be settled according to Theorem 5 or case 1 applies.

" $\leftarrow$ " :

Let s-t be an optimal query and $\{s = v_0, v_1, \ldots, v_r = t\}$ be the vertices on the path from s to t.

Assume that the s-t-path is not one of the cases 1, 2 or 3. Then, t is not on the path from s to l and only the following cases remain possible:

(i) t is in a subtree of s

(ii) t and s are connected through vertex $c \neq l$ with degree ≥ 3.

(iii) t and s are connected through vertex $c = l$ with degree ≥ 2.

We will show that all three cases result in a contradiction.

(i) t is in a subtree of s

If $\{v_1, \ldots, v_{r-1}\}$ are vertices with degree ≤ 2, then case 3 applies for s-t. Hence at least one vertex in $\{v_0, \ldots, v_{r-1}\}$ has degree > 2. Let v be this vertex, then:

$$\begin{aligned} \text{cost}(v) &= \text{dist}(s,v) + |\,\text{dist}(v,l) - \text{dist}(t,l)| \\ &\overset{*}{=} \text{dist}(s,v) - \text{dist}(v,l) + \text{dist}(t,l) \\ &= \text{dist}(s,v) + \text{dist}(v,t) \\ &= \text{dist}(s,t) \end{aligned} \tag{3.9}$$

where $\overset{*}{=}$ holds, since $\text{dist}(t,l) > \text{dist}(v,l)$. v has at least 3 adjacent vertices, and at least one vertex u which is not on the s-t-path with

$$\begin{aligned} \text{cost}(u) &= \text{dist}(s,u) + |\,\text{dist}(u,l) - \text{dist}(t,l)| \\ &= \text{dist}(s,v) + 1 - |\,\text{dist}(v,l) + 1 - \text{dist}(t,l)| \\ &= \text{dist}(s,v) + 1 + - \text{dist}(v,l) - 1 + \text{dist}(t,l) \\ &= \text{dist}(s,v) + \text{dist}(v,t) \\ &= \text{dist}(s,t) \end{aligned} \tag{3.10}$$

Since v is on the s-t-path, v must be settled and therefore u must be settled too, though it is not on the s-t-path. Hence the search space of the s-t-query is not optimal.

(ii) t and s are connected through vertex $c \neq l$ with degree ≥ 3.

If v_0 has degree 1 and $\{v_1, \ldots, v_{r-1}\}$ has only one vertex with degree 3, then case 2 applies. Otherwise s has degree > 1 or $\{v_1, \ldots, v_{r-1}\}$ has at least one vertex with degree > 3 or more than one vertex with degree 3.

Let $\mathcal{C}^l_{s,t}$ be the core of the s-t-query, i.e. the intersection of the path from s to l and the path from s to t. Since t is not in the path q from s to l, ALT searches all vertices within $\text{dist}(t, \mathcal{C}^l_{s,t}) \geq 1$ of the core of the s-t-path. Hence it is sufficient to show that in each remaining case a vertex u exists with $\text{dist}(\mathcal{C}^l_{s,t}, u) \leq \text{dist}(\mathcal{C}^l_{s,t}, t)$.

If $\{v_1, \ldots, v_{r-1}\}$ has only one connecting vertex with degree 3 but v_0 has a degree > 2, v_0 is obviously in the core and has one adjacent vertex u which is not on the s-t-path. For u it holds that $\text{dist}(\mathcal{C}^l_{s,t}, u) = 1 \leq \text{dist}(\mathcal{C}^l_{s,t}, t)$ since the target has at least distance 1 from the core. Contradiction since u is settled from ALT and not on the s-t-path.

If $\{v_1, \ldots, v_{r-1}\}$ has at least one vertex v with degree > 3, then this vertex has at least one adjacent vertex u which is not on the s-t-path and not on the s-l-path for which $\text{dist}(\mathcal{C}^l_{s,t}, u) = 1 \leq \text{dist}(\mathcal{C}^l_{s,t}, t)$. Contradiction since u is settled from ALT and not on the s-t-path.

If $\{v_1, \ldots, v_{r-1}\}$ has more than one vertex with degree 3, then let v be the vertex which has one adjacent vertex u which is not on the s-t-path and not on the s-l-path (at least one must exist if more than one vertex with degree ≥ 3 is on the path). As in the other cases, this vertex u is in the ALT search space though not on the s-t-path which results in an contradiction.

(iii) t and s are connected through vertex $c = l$.

If s has degree 1 and the vertices $\{v_1, \ldots, v_{r-1}\}$ have degree 2, case 2 applies. Otherwise s has degree > 1 or $\{v_1, \ldots, v_{r-1}\}$ has at least one vertex with degree > 2. As in case (ii) the target t is not in the core $\mathcal{C}^l_{s,t}$ of the s-t-query and hence it is sufficient to show that one vertex u exists which is $\text{dist}(\mathcal{C}^l_{s,t}, u) \leq 1 \leq \text{dist}(\mathcal{C}^l_{s,t}, t)$

If s has degree > 1 the to s adjacent vertex u has $\text{dist}(\mathcal{C}^l_{s,t}, u) = 1 \leq \text{dist}(\mathcal{C}^l_{s,t}, t)$. If in $\{v_1, \ldots, v_{r-1}\}$ at least one vertex has degree > 2, then this vertex has an adjacent vertex u which is not on the s-t-path with $\text{dist}(\mathcal{C}^l_{s,t}, u) = 1 \leq \text{dist}(p\mathcal{C}^l_{s,t}, t)$. In both cases u is in the search space and the search space is not optimal. Contradiction.

□

With this knowledge, we can compute for each vertex, which s-t-paths are optimal if this vertex is a landmark. By solving a set-covering problem we can then compute an upper bound on the smallest number of landmarks which are needed to obtain an optimal search space for all queries. Note that we could also use any known approximation algorithm for set-covering. This approach does not give the exact bound, since some s-t-paths which are not optimal for one landmark can be optimal if another landmark (for which alone this s-t-path is not optimal, too) is added.

3.3.5 On paths between landmarks

In this section we will examine the vertices on paths between two (or more) landmarks on trees. We will show that adding to the landmark set any vertex which is on a path between some other landmarks, does not yield a reduction of the search space.

Theorem 7. *Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a tree and $\mathbf{L} = \{l_1, \ldots, l_k\}$ with $k \geq 2$ a set of landmarks. W.l.o.g. be v on a path between l_1 and l_2. Then, the set $\mathbf{L}' = \{l_1, \ldots, l_k, v\}$ will not improve the overall ALT search space.*

Proof. Suppose the set $\mathbf{L}'$ improves the resulting overall ALT search space, then there must be one s-t-query q which is improved by the landmark v. Then, q must be improved by one of the ways stated in Section 3.3.1 (i) and 3.3.2 (ii).

We will first consider case (i): The search space can only be improved in this case if the target t is closer to the core of the s-t-query than before. The core of the s-t-query is the intersection of the s-t-path and one of the paths from the source to the landmarks. Since v is on a path between two landmarks, the core of the s-t-query can not be enlarged, since no more vertices than before can be in the intersection. Contradiction.

Consider now (ii): Then, the landmark v must exclude a subtree from the ALT search space, which is searched by the ALT algorithm if it only uses the set $\mathbf{L}$ as landmarks. From the s-t-path, only vertices which are not in the direction of one of the landmarks in $\mathbf{L}$ may be settled using $\mathbf{L}$, therefore v must exclude at least one vertex from the search space which is not in direction on another landmark but in direction of v. Since v is on the path between l_1 and l_2 this is not possible. If a vertex can be excluded from the search space using v it has already be excluded by l_1 or l_2. Contradiction.

Hence the overall search space with the set $\mathbf{L}'$ is the same as with $\mathbf{L}$. □

With this proof we get the intuition, that in a tree, landmarks should only be placed in leaves. This intuition is correct, as the following corollary states.

Corollary. *Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a tree. Then only leaves or the root if it has only one child, i.e. vertices with only one adjacent vertex, are in the optimal set of k landmarks for $k \geq 2$.*

Proof. We assume $\{l_1, \ldots, l_k\}$ with $k \geq 2$ are optimal landmarks and let w.l.o.g l_1 have more than one adjacent vertex. If none of the adjacent vertices is on a path from l_1 to another landmark, then according to Theorem 7 l_1 has no more effect since it is on at least one path between two other landmarks. Hence let u be the adjacent vertex which is not on the path from l_1 to any other landmark. Then, the set $\{u, l_2, \ldots, l_k\}$ is at least as good as $\{l_1, \ldots, l_k\}$. □

In Figure 3.3 we illustrated how one can draw a tree so that one can easily see the distance from the target t to the source-landmark-path s-l for an s-t-query as well as which s-t-queries are yet optimal. Originating from the idea behind this illustration that all vertices between landmarks act as if they were landmarks which has been shown in Theorem 7 we will introduce a new heuristic for the selection of a set of landmarks in Section 4.4.

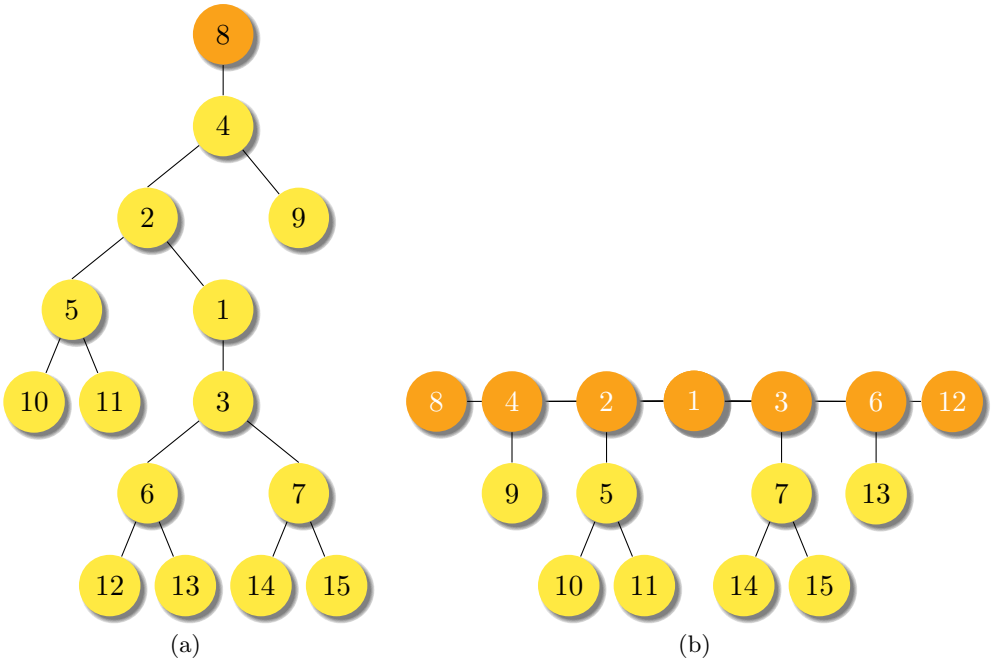

Figure 3.3: In both images we see the same complete binary tree with 15 vertices (in standard notation, 1 would be the root). On the left (a) the tree is drawn with one landmark vertex 8 as root and on the right (b) the path from the landmark 8 to the landmark 12 substitutes a root vertex.

4. Implementations and algorithms

In this Chapter we will first give another interpretation of the problem to select the landmarks, which will lead us to a binary linear program formulation. Afterwards we will give two formulations of a greedy algorithm which holds an approximation guarantee of $\frac{e-1}{e}$ of the optimal solution. Then we will introduce a new heuristic and finally give a brute force algorithm to compute a optimal set of k landmarks.

4.1 Selecting landmarks as maximum coverage problem

In contrast to the problem MinALT defined in Chapter 3 we will first define the problem of maximum coverage (MaxCover) in this section and afterwards show that the problem MinALT can be reduced to the problem MaxCover.

Definition 4.1 (MaxCover)**.** Given a set of elements $\{e_1, \ldots, e_r\}$, a collection $\mathcal{S} = \{S_1, \ldots, S_n\}$ of sets where $S_i \subset \{e_1, \ldots, e_r\}$ and an integer k, the problem MaxCover is to find a subset $\mathcal{S}' \subset \mathcal{S}$ with $|\mathcal{S}'| \leq k$ so that a maximum number of elements e_i are covered: $\bigcup_{S_i \in \mathcal{S}'} S_i$ is maximal.
Such a set $\mathcal{S}'$ is called a set with *maximum coverage*.

The idea of interpreting the problem of selecting k landmarks as a maximum coverage problem is to compute, for each vertex v and each s-t-query, which vertices the vertex v would exclude from the Dijkstra search if it would be a landmark. Therefore we start by computing the Dijkstra search space for each s-t-query and setting up a $|V|^3 \times |V|$ matrix with one column for each vertex in every Dijkstra search space and one row for each vertex in $\mathbf{V}$, interpreted as potential landmark. Each column is assigned to one vertex v in the Dijkstra search space of an s-t-query and each row to one potential landmark l_i as illustrated in Figure 4.1. Each entry of the matrix is initialized with 0.

Then, for each vertex v in the Dijkstra search space of an s-t-query we compute for every potential landmark l_i if the ALT algorithm would visit v on the s-t-query when using l_i as landmark. If the ALT algorithm would not visit v using l_i (i.e. l_i would exclude v from the search space), we set the entry for row l_i in the column assigned to the vertex v to 1. For this matrix, selecting k rows so that a maximum number of columns are covered is equivalent to the maximum coverage problem.

Theorem 8. *There is an polynomial reduction from the problem* MinALT *to* MaxCover *with complexity in* $\mathcal{O}(|V|^4)$.

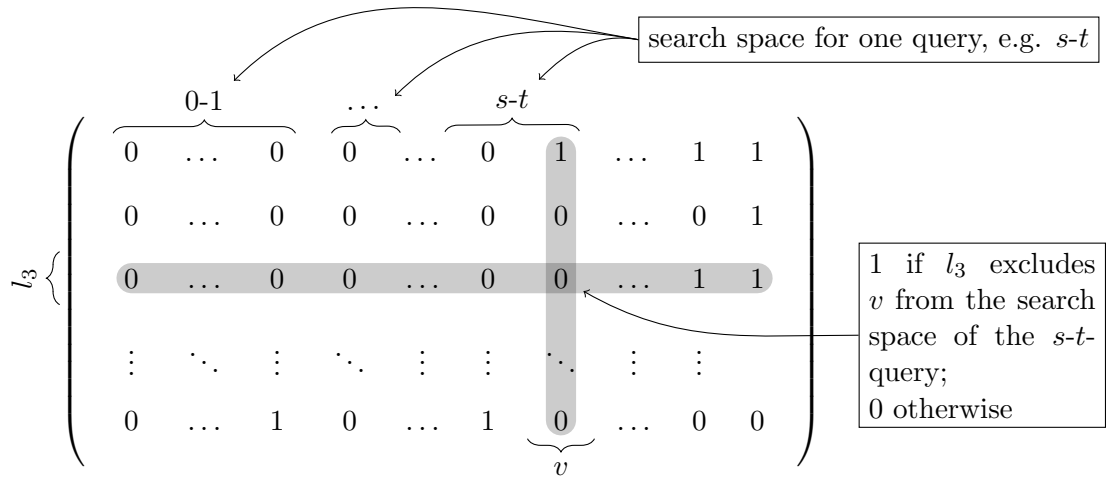

Figure 4.1: The matrix for the landmark selection problem interpreted as maximum coverage problem

Proof. Let $\{e_1, \ldots, e_r\} = \{z_v(s,t) | v, s, t \in \mathbf{V} \text{ with } \text{dist}(s,v) \leq \text{dist}(s,t)\}$ be the set of elements. Every element $z_v(s,t)$ represents one vertex v in an s-t-query and equals one entry in the matrix. For each potential landmark l_i (for $i = 1, \ldots, |\mathbf{V}|$) we introduce a set $S_i \subset \{z_v(s,t) | v, s, t \in \mathbf{V} \text{ with } \text{dist}(s,v) \leq \text{dist}(s,t)\}$, such that $z_v(s,t) \in S_i$ if the vertex v is excluded from the Dijkstra s-t-search-space-model if l_i is a landmark, each $z_v(s,t) \in S_i$ equals one 1 in the row i of the matrix.

Claim. *A set $\mathcal{S}' \subset \{S_1, S_2, \ldots, S_n\}$ with $|\mathcal{S}'| \leq k$ is a optimal solution for the problem* MaxCover *if and only if $\mathbf{L} = \{l_i | S_i \in \mathcal{S}'\}$ is optimal solution for the problem* MinALT.

$\Rightarrow$ **:** Let $\mathcal{S}' \subset \{S_1, S_2, \ldots, S_n\}$ with $|\mathcal{S}'| \leq k$ be a optimal solution for the problem MaxCover.

Assume that we can find a set $\widetilde{\mathbf{L}}$ of k landmarks such that the search space is smaller than with the set $\mathbf{L} = \{l_i | S_i \in \mathcal{S}'\}$ induced by the solution of the problem MaxCover. Then these landmarks must exclude more vertices from the original Dijkstra search spaces than the landmarks in $\mathbf{L}$ do. Therefore the set $\widetilde{\mathcal{S}}' = \{S_i | l_i \in \widetilde{\mathbf{L}}\}$ covers more elements than $\mathcal{S}'$. Contradiction to the optimality of the solution $\mathcal{S}'$.

$\Leftarrow$ **:** Let $\mathbf{L} = \{l_1, \ldots, l_k\}$ be an optimal solution of the problem MinALT.

Assume that we can find a set $\widetilde{\mathcal{S}}'$ with $|\widetilde{\mathcal{S}}'| \leq k$ such that it covers more elements e_j than the set $\mathcal{S}' = \{S_i | l_i \in \widetilde{\mathbf{L}}\}$ induced by the solution of the problem MinALT. This solution $\widetilde{\mathcal{S}}'$ must cover more elements and therefore the set $\widetilde{\mathbf{L}} = \{l_i | S_i \in \widetilde{\mathcal{S}}'\}$ must also exclude more elements from the Dijkstra search space than the set $\mathbf{L}$. Contradiction to the optimality of the solution $\mathbf{L}$.

This proves the claim and therefore it remains to show the complexity of the reduction. For the reduction we need to compute the matrix with $|\mathbf{V}|^4$ $z_v(s,t)$ elements as entries and then compute $|\mathbf{V}|$ subsets of the set $\{z_v(s,t) | v, s, t \in \mathbf{V} \text{ with } \text{dist}(s,v) \leq \text{dist}(s,t)\}$. To compute the matrix $\mathcal{O}(|\mathbf{V}|^4)$ is needed since for each potential landmark and each vertex v on a s-t path it must be computed ($\mathcal{O}(1)$) whether it is in the ALT search space or not for this landmark. Afterwards the sets must be computed out of the rows of the matrix which again is an $\mathcal{O}(|\mathbf{V}|^4)$ operation. Hence the reduction is in $\mathcal{O}(|\mathbf{V}|^4)$. □

Note that we can consider the landmarks independently (as we do it in the proof of the theorem) for the search space model used in this thesis. This is possible, since it depends

solely on the distance and the potential of a vertex whether it is in the search space of an given s-t-query or not. For other models this is not necessarily possible as we showed in an example in Section 3.1 that the actual ALT search space increased for this example even though we added one more landmark.

4.2 Integer linear program

For the problem MinALT of selecting k landmarks so that the search space of the ALT algorithm is minimal, we can formulate a integer linear program by using the formulation of the problem as a maximum coverage problem MaxCover. In this chapter we will give a binary linear program using the search space model used throughout this thesis and introduced in Section 3.1.

This approach models for each vertex v in any Dijkstra search space, which landmarks exclude this vertex v from the ALT search space. With the linear program, the set of k landmarks which excludes most vertices from the ALT search space is computed. Before we can formulate the binary linear program, we must preprocess the distances between all vertices in the graph. After that, let $\mathbf{V}(s,t) := \{v \in V \,|\, \text{dist}(s,v) \leq \text{dist}(s,t)\}$ denote the search space model of an s-t-query using Dijkstra's algorithm. These search spaces have to be precomputed for every pair of vertices s and $t \in \mathbf{V}$. For s,$t \in \mathbf{V}$ and $v \in \mathbf{V}(s,t)$ we use the variables

$$z_v(s,t) = \begin{cases} 0 & v \text{ is not in the ALT search space model of the } s\text{-}t\text{-query} \\ 1 & \text{otherwise} \end{cases} \tag{4.1}$$

and variables $\mathbf{L}(v)$ being 1 if v is a landmark and 0 otherwise. Finally we let $\mathbf{L}_v(s,t)$ denote the set of landmarks l such that v is not in the ALT search-space-model of the s-t-query if landmark l is used. This set can be computed using only the precomputed distances and the resulting potentials.

With these variables the binary linear program can be formulated as the following maximization problem.

$$\max \sum_{s,t \in \mathbf{V}, v \in \mathbf{V}(s,t)} z_v(s,t) \tag{4.2}$$

subject to

$$z_v(s,t) \in [0,1] \qquad \text{for } s,t \in \mathbf{V} \text{ and } v \in \mathbf{V}(s,t) \tag{4.3}$$

$$\mathbf{L}(v) \in \{0,1\} \qquad \text{for } v \in \mathbf{V} \tag{4.4}$$

$$z_v(s,t) \leq \sum_{l \in \mathbf{L}_v(s,t)} \mathbf{L}(l) \qquad \text{for } s,t \in \mathbf{V} \text{ and } v \in \mathbf{V}(s,t) \tag{4.5}$$

$$\sum_{v \in \mathbf{V}} \mathbf{L}(v) \leq k \tag{4.6}$$

In this linear programming formulation, we maximize over $z_v(s,t)$ for all vertices v in the Dijkstra search space for all pairs (s,t). This implies a minimal overall search space of the ALT-algorithm, since the overall search space of the ALT algorithm is the overall Dijkstra search space minus $\sum_{s,t \in \mathbf{V}, v \in \mathbf{V}(s,t)} z_v(s,t)$.

It is sufficient to require, that the variable $z_v(s,t)$ must be in the interval from 0 to 1, since integrality of $z_v(s,t)$ follows from the fact that we are maximizing the variables $z_v(s,t)$ and that they only appear in constraint (4.5) which has an integral right hand side if $\mathbf{L}(l)$

is integral. Finally, constraint (4.6) requires the number of landmarks be smaller or equal to k.

A downside of this formulation is that this linear program is quite memory-consuming. To represent $z_v(s,t)$ for each vertex v in the Dijkstra search space model in every s-t-pair, $|\mathbf{V}|^3$ variables are needed. For the set $\mathbf{L}_v(s,t)$ we need even $\mathcal{O}(|V|^4)$ variables since for each vertex v in the Dijkstra search space model in every s-t-pair a set of landmarks must be stored. Additionally we need to precompute $|\mathbf{V}| * |\mathbf{V}|$ distances, which is computationally in the complexity of $\mathcal{O}(|V|^3)$ (see Algorithm 4.1). The potential function can be computed out of the distances when needed since it does only depend on s, t, v and l whether l is in $\mathbf{L}_v(s,t)$ or not.

4.3 The greedy algorithm

In order to describe the greedy algorithm, we will first describe two methods which are used in the greedy algorithm as well as for the brute force algorithm in Section 4.5.

The Algorithm 4.1 computes all-pair-shortest-paths, i.e. the distances between all pairs of vertices, and is called Floyd-Warshall algorithm (Cormen et al., 2001). The complexity of this algorithm is $\mathcal{O}(|\mathbf{V}|^3)$.

Algorithm 4.1 FLOYD-WARSHALL()

Require: A graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with positive length function len
Ensure: $\text{dist}(u,v)$ for all $u, v \in \mathbf{V}$
1: **for all** $u \in \mathbf{V}$ **do**
2: **for all** $v \in \mathbf{V}$ **do**
3: **if** $(u,v) \in \mathbf{E}$ **then**
4: $\text{dist}(u,v) \leftarrow \text{len}(u,v)$
5: **else**
6: $\text{dist}(u,v) \leftarrow \infty$
7: **for** $k = 1 \ldots |\mathbf{V}|$ **do**
8: **for all** $s \in \mathbf{V}$ **do**
9: **for all** $t \in \mathbf{V}$ **do**
10: $\text{dist}(s,t) \leftarrow \min\{\text{dist}(s,t), \text{dist}(s,k) + \text{dist}(k,t)\}$

To compute the search space for one pair of source and target vertices s and t the Algorithm 4.2 is used. It requires precomputed distances between all pairs of vertices in $\mathbf{V}$ and computes the search space according to the model given in Section 3.1. The algorithm has a complexity of $\mathcal{O}((|\mathbf{V}| \cdot k) + |\mathbf{V}|) = \mathcal{O}(|\mathbf{V}| \cdot k)$

Algorithm 4.2 COMPUTE_$\mathbf{V_L}(s,t)(\mathbf{L}, s, t)$

Require: $\text{dist}(u,v)$ for all $u, v \in \mathbf{V}$, set of landmarks $\mathbf{L}$, $s, t \in \mathbf{V}$
Ensure: correct set $\overline{\mathbf{V}}_\mathbf{L}(s,t)$ of vertices in the search space of the s-t-query
1: **for all** $v \in \mathbf{V}$ **do**
2: $\pi_t^\mathbf{L}(v) \leftarrow \max_{l \in \mathbf{L}} \{\text{dist}(v,l) - \text{dist}(t,l), \text{dist}(l,t) - \text{dist}(l,v)\}$
3: **for all** $v \in \mathbf{V}$ **do**
4: **if** $\text{dist}(s,v) + \pi_t^\mathbf{L}(v) \leq \text{dist}(s,t)$ **then**
5: $\overline{\mathbf{V}}_\mathbf{L}(s,t) \leftarrow \overline{\mathbf{V}}_\mathbf{L}(s,t) \cup \{v\}$

The greedy algorithm as it is given in Algorithm 4.3 computes the one vertex v which minimizes the search space if added to the so far computed set $\mathbf{L}$ of landmarks in each iteration. This vertex v is then added to the set $\mathbf{L}$ of landmarks. After k iterations $\mathbf{L}$ consists of k landmarks and is returned as the set of selected landmarks.

Algorithm 4.3 GREEDY(k)

Require: Integer k
1: $\mathbf{L} \leftarrow \emptyset$
2: Floyd-Warshall() {compute all-pair-shortest-paths}
3: **for all** $i = 1, \ldots, k$ **do**
4: **for all** $l \in (\mathbf{V} - \mathbf{L})$ **do**
5: **for all** $(s, t) \in \mathbf{V}$ **do**
6: compute_$\mathbf{V_L}(s, t)(\mathbf{L}, s, t)$ {compute search space}
7: sum$[l] = \sum_{s,t \in \mathbf{V}} \overline{\mathbf{V}}_{\mathbf{L}}(s, t)$
8: bestLM $\leftarrow$ argmin{sum$[l] \mid l \in (\mathbf{V} - \mathbf{L})$}
9: $\mathbf{L} \leftarrow \mathbf{L} \cap$ {bestLM}
10: **return** Set $\mathbf{L}$ of k landmarks

We can also formulate this greedy algorithm in a similar way as the integer linear program in Section 4.2. We no longer add the one vertex v to the set $\mathbf{L}$ of landmarks which minimizes the resulting search space but the one which excludes most vertices from the search space resulting from the so far computed set of landmarks. Since these formulations are equivalent both variants always choose the same vertices and hence result in the same set of landmarks.

Algorithm 4.4 ALTERNATIVEGREEDY(k)

Require: Integer k
1: $\mathbf{L} \leftarrow \emptyset$
2: Floyd-Warshall() {compute all-pair-shortest-paths}
3: **for all** $v \in \mathbf{V}$ **do**
4: sum$[v] \leftarrow 0$
5: min $\leftarrow \infty$
6: minLm $\leftarrow$ null
7: **for all** $i = 1, \ldots, k$ **do**
8: **for all** $s \in \mathbf{V}$ **do**
9: **for all** $t \in \mathbf{V}$ **do**
10: **for all** $v \in \mathbf{V}$ **do**
11: **if** dist$(s, v) \leq$ dist(s, t) **then**
12: **for all** $l \in \mathbf{V}$ **do**
13: potential $\leftarrow \max\{\text{dist}(v, l) - \text{dist}(t, l), \text{dist}(l, t) - \text{dist}(l, v)\}$
14: **if** dist(s, v) + potential $\leq$ dist(s, t) **then**
15: sum$[l] \leftarrow$ sum$[l] + 1$
16: **for all** $l \in \mathbf{V}$ **do**
17: **if** sum$[l] <$ min **then**
18: min $\leftarrow$ sum$[l]$
19: minLm $\leftarrow l$
20: $\mathbf{L} \leftarrow \mathbf{L} \cup \{l\}$
21: **return** Set $\mathbf{L}$ of k landmarks

With the alternative formulation of the greedy algorithm as it is given in Algorithm 4.4 we are able to interpret the algorithm as a greedy algorithm for the maximum coverage problem. This problem is well known in literature and provides us with an approximation guarantee of $(1 - \frac{e}{1}) = \frac{e-1}{e}$ of the optimal solution for finding a set which excludes a maximum number of vertices (Hochbaum, 1997), where e is Euler's number.

Theorem 9 ((Hochbaum, 1997))**.** *Let* weight($GREEDY$) *be the weight of the solution of the greedy algorithm for the problem* MAXCOVER *and* weight(OPT) *be the weight of the*

optimal solution for the same problem, then:

$$\text{weight}(GREEDY) \geq [1 - (1 - \frac{1}{k})^k] \cdot \text{weight}(OPT) > (1 - \frac{1}{e}) \cdot \text{weight}(OPT) \quad (4.7)$$

Corollary. *Let* coverage($GREEDY$) *be the number of vertices which can be excluded from the Dijkstra search space in the greedy solution for the* MinALT *problem and* coverage(OPT) *be the number of vertices which can be excluded in the optimal solution of the same problem, then:*

$$\text{coverage}(GREEDY) \geq [1 - (1 - \frac{1}{k})^k] \cdot \text{coverage}(OPT) > (1 - \frac{1}{e}) \cdot \text{coverage}(OPT) \quad (4.8)$$

Proof. The corollary follows directly from Theorem 9, since the equality of the problems MaxCover and MinALT is shown in Theorem 8. □

We also know that this is basically best we can do with the maximum coverage approach, since an approximation threshold of $\frac{e-1}{e}$ (up to low-order terms) is shown for the maximum coverage problem (Feige, 1998).

4.4 A new heuristic

The greedy algorithm presented in the previous section may perform well, but its complexity as well as its memory requirements are high and therefore the algorithm will not be applicable on real world data. Hence we will give another heuristic which has been inspired by our understanding of the effects of landmarks on trees gained in Section 3.3.

As before, we will select one landmark at a time and therefore have k iterations to select k landmarks.

For the first iteration, we temporarily add a random vertex to the set of landmarks which will be deleted after the first iteration. In each iteration we first add a dummy vertex r to $\mathbf{V}$ and edges (r, l) to $\mathbf{E}$ with length zero from this vertex r to the vertices l in the set $\mathbf{L}$ of so far determined landmarks. We then build a search tree with the dummy vertex r as root. In this search tree, we calculate a weight

$$\text{weight}(v) = \text{dist}(r, v) + \sum_{w \text{ child of } v} \text{weight}(v) \quad (4.9)$$

which is recursively computed in Algorithm 4.6 beginning at the root vertex. Then the algorithm descends from the root vertex r always to the child with the highest weight($\cdot$) value until a leaf is reached. This leaf is selected and added to the set of landmarks as shown in Algorithm 4.7. In the first iteration, the random vertex added to the set of landmarks before the iteration gets deleted again. Note that this way the algorithm is usually not selecting the node with the highest weight($\cdot$) value.

Unfortunately this heuristic could not be tested due to time limitations. In our opinion, the following improvements are interesting:

First, one minor improvement should be to start with a leaf vertex instead of a completely randomly determined vertex. An major improvement may be to introduce another set $\widetilde{\mathbf{L}}$ to store all vertices on a shortest path between two landmarks (including the landmarks itself) and then connect the dummy root vertex r to all vertices in the set $\widetilde{\mathbf{L}}$ with edges of zero length. This variant is inspired by the fact that in trees vertices on the path between landmarks should not be added to the set of landmarks as it is stated in Section 3.3.5.

Algorithm 4.5 WEIGHTEDPATHS(k)

Require: graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and Integer k
1: x ← random vertex
2: $\mathbf{L} \leftarrow \{\text{x}\}$
3: **for** i = 1 ... k **do**
4: $\mathbf{V} \leftarrow \mathbf{V} \cup \{$ root $\}$ {add an dummy vertex root}
5: **for all** $l \in \mathbf{L}$ **do**
6: $\mathbf{E} \leftarrow \mathbf{E} \cup \{(\text{root},l)\}$ {and edges with length 0 from root to all vertices in $\mathbf{L}$}
7: len(root,l) ← 0
8: Dijkstra($(\mathbf{V}, \mathbf{E})$,root) {calculate dist(root,v) for all $v \in \mathbf{V}$}
9: calculateWeights(root)
10: l ← maxWeightLeaf(root)
11: $\mathbf{L} \leftarrow \mathbf{L} \cup \{l\}$
12: **if** i = 1 **then**
13: $\mathbf{L} \leftarrow \{l\}$ {delete random vertex x}
14: **return** Set $\mathbf{L}$ of k landmarks

Algorithm 4.6 CALCULATEWEIGHTS(v)

Require: Vertex v
1: weight(v) ← dist(root,v)
2: **for all** child w of v **do**
3: weight(v) ← weight(v)+calculateWeights(w)
4: **return** weight(v)

Algorithm 4.7 MAXWEIGHTLEAF(v)

Require: Vertex v
1: max ← 0
2: maxVertex ← null
3: **for all** child w of v **do**
4: **if** weight(w) > max **then**
5: max ← weight(w)
6: maxVertex ← w
7: **if** maxVertex = null **then**
8: **return** v
9: **return** maxWeightLeaf(maxVertex)

4.5 The brute force algorithm

The brute force algorithm to solve the landmark selection problem given in Algorithm 4.8 is similar to the greedy algorithm given in Algorithm 4.3, except for that we do not calculate the resulting search space $k \cdot \mathbf{V}$ times but $\binom{|\mathbf{V}|}{k}$ times: Instead of computing the next best landmark with smallest resulting search space for k iterations in the greedy algorithm we compute the set $\mathbf{L} \subset \mathbf{V}$ of k landmarks (of which $\binom{|\mathbf{V}|}{k}$ different exist) with the smallest resulting search space. As in the greedy algorithm we use Algorithm 4.1 to compute all-pair-shortest-paths and Algorithm 4.2 to compute the search space for the s-t-query using a set $\mathbf{L}$ of landmarks.

Algorithm 4.8 BruteForceLandmarks(k)

Require: Integer k
1: Floyd-Warshall()
2: $\mathbf{L}_{min} \leftarrow \emptyset$
3: min $\leftarrow \infty$
4: **for all** subsets $\mathbf{L} \subset \mathbf{V}$ with $|\mathbf{L}| = k$ **do**
5: **for all** $(s,t) \in \mathbf{V}$ **do**
6: compute_$\mathbf{V}_\mathbf{L}(s,t)(\mathbf{L},s,t)$
7: sum $\leftarrow \sum_{s,t \in \mathbf{V}} \overline{\mathbf{V}}_\mathbf{L}(s,t)$
8: **if** sum $<$ min **then**
9: min $\leftarrow$ sum
10: $\mathbf{L}_{min} \leftarrow \mathbf{L}$
11: **return** $\mathbf{L}_{min}$ of optimal landmarks

Since $\binom{|\mathbf{V}|}{k} \in \theta(|\mathbf{V}|^k)$, the algorithm has a complexity of $\mathcal{O}(|\mathbf{V}|^3 + |\mathbf{V}|^k \cdot k \cdot (|\mathbf{V}|^2 \cdot |\mathbf{V}| \cdot k + |\mathbf{V}|^2)) = \mathcal{O}(|\mathbf{V}|^k \cdot k \cdot |\mathbf{V}|^2 \cdot |\mathbf{V}| \cdot k) = \mathcal{O}(|\mathbf{V}|^{k+3} \cdot k^2)$. Obviously, an algorithm with this complexity is not able to handle large graphs, let alone real world road networks.

5. Experiments

We were provided with an existing C++ code which implements the heuristics used so far like *random*, *avoid*, *advanced avoid* and *maxCover* from the Algorithmics I group at the Institute of Theoretical Informatics at the Karlsruhe Institute of Technology. Additionally we implemented the greedy algorithm 4.3 from Section 4.3 as well as a method to calculate the resulting search space which has been used to measure the quality of a set of landmark. This method basically calls algorithm 4.2 for each pair s,$t \in \mathbf{V}$ and hence measures the quality way more precisely than the reduced costs method introduced by Goldberg (Goldberg and Werneck, 2005) - though unfortunately it is only applicable for small graphs due to its complexity. The integer linear program from Section 4.2 has been implemented in Java using CPLEX (ILOG, 2009) as solver for the integer linear program.

The hardware environment for the experiments was a server with two Intel Xeon E5340 Quad-Core CPUs with 2.66Ghz per core and 6MB of cache per CPU. The server is equipped with 32GB of memory. All implementations did only use one core while running, so that several instances of the same or different programs ran simultaneously. Regarding the software environment the server is running SUSE Linux 11.0 as operating system with kernel 2.6.25.20-0.5-default. The java virtual machine is running at version 1.6.0_17 and the implementation in C++ has been compiled using gcc version 4.3.3 with parameters '-O4 -funroll-loops -DNDEBUG -DUNPACK'.

We were eager to try the methods on graphs as similar as possible to real world conditions. Due to the complexity and hence the resulting runtime and memory requirements for real world graphs, we had to limit ourself to small parts of these real world graphs. We gratefully acknowledge the support of PTV AG, Karlsruhe for providing us with real world road network graphs, though we were not able to use the whole graphs for the experiments. Instead we extracted strongly connected components from these graphs to obtain graphs of manageable sizes which are most likely connected.

In the following sections we will compare the heuristics used so far with the greedy algorithm as well as the integer linear program. Therefore we will first give a brief overview on how the different heuristics work:
random picks the amount of landmarks required as the set of landmarks randomly.
avoid first grows a shortest path tree T_r with random root r in which for each iteration for each vertex v the *weight* as difference between $\text{dist}(r, v)$ and the lower bound induced by the set $\mathbf{L}$ of so far selected landmarks. Then, the *size* of each vertex v is computed as the sum of weight of the vertices in the subtree of v if there is no landmark in the subtree.

If a landmark is in the subtree of v, the size of v is 0. Finally the algorithm traverses the tree starting at the vertex w with maximum size always to the child with the largest size until a leaf is reached. This leaf is added to the set of landmarks.
advanced avoid is a variant of avoid which tries to compensate the main disadvantage of avoid by exchanging the first landmarks selected by avoid by other landmarks later generated by avoid.
maxCover also tries to compensate the disadvantages of avoid by computing a set of landmarks 4 times bigger than needed using avoid. Out of these landmark candidates maxCover selects the landmarks then through several attempts using a local search routine.

5.1 Comparison with the Integer Linear Program

Despite the progress linear program solver like CPLEX made in the past years, we have not been able to apply our integer linear program to graphs bigger than around 125 to 150 vertices when requiring the solver to finish within 6 hours. We may note that though the linear program is quite memory consuming (for 150 vertices our linear program uses about 10GB of main memory) this was not yet the limiting factor since even within the time limit set to 6 hours there has been instances with at most 125 or 150 vertices which CPLEX has not been able to solve optimally. For these instances no solution or if available the best feasible solution is given.

Since it is obviously not possible to test the linear program on real world graphs, we limit ourself to a comparison of how good the solution of the heuristics used so far and the greedy implementation is compared to the optimal solution computed by CPLEX using the integer linear program.

The search space in the following Table 5.1 has been computed without considering the s-t-queries with $\text{dist}(s,t) = \infty$ at all, i.e. the search space of queries such that the target t is not reachable from the source s is not taken into account.

In the comparison in Table 5.1 we can see, that the quality of the generated landmarks improves with the time needed to compute them. Overall we can say that the quality of the generated set of landmarks usually is in the following order

$$\text{adv. avoid} \leq \text{avoid} \leq \text{maxCover} \leq \text{greedy} \leq \text{ILP}$$

If we consider the time needed to generate the landmarks, we observe almost the reserve order except for adv. avoid which is usually taking more time than avoid. On our graphs we can not see an improvement from advanced avoid over avoid, for most graphs we tested, advanced avoid has been worse than avoid.

We can also see that for most of our combination of graphs and generated landmarks the greedy algorithm is as good as the optimal algorithm, though we suppose that for bigger graphs the gap between them is getting clearer (as we can see a first gap at the lux-graph with 100 vertices).

5.2 Comparing the Greedy Algorithm

Even though the greedy algorithm is less complex than the brute force algorithm as well as is requires less memory than the integer linear program, once again we are not able to run this algorithm on real-world road-network graphs. On our configuration, the implementation of the greedy algorithm could at most be applied to compute around 12 to 16 landmarks on a graph with 1500 vertices without exceeding the time limitations of 90 hours.

GRAPHS	DEU		LUX	
	TIME	SEARCH SPACE	TIME	SEARCH SPACE
50 vertices				
avoid	0.0001 s	21,643	0.0001 s	26,897
adv. avoid	0.0002 s	21,785	0.0003 s	27,945
maxCover	0.0006 s	21,209	0.0005 s	26,396
greedy	0.07814 s	20,689	0.1036 s	26,309
ilp	33.420 s	20,689	34.548 s	26,309
100 vertices				
avoid	0.0003 s	167,877	0.0003 s	179,275
adv. avoid	0.0007 s	174,042	0.0009 s	188,275
maxCover	0.001 s	156,353	0.00111 s	179,055
greedy	1.6513 s	156,043	1.6618 s	178,359
ilp	2h 4m 42s	156,043	10h 30m 47s	178,146
125 vertices				
avoid	0.0005 s	278,486	0.0003 s	334,616
adv. avoid	0.0008 s	291,885	0.0011 s	350,209
maxCover	0.0013 s	263,089	0.0016 s	317,208
greedy	3.4267 s	262,647	3.6162 s	308,232
ilp	6h 1m 54s	- (*)	6h 1m 54s	- (*)

Table 5.1: Comparison of the heuristics used so far with the greedy algorithm and the integer linear program solved with CPLEX on two different graphs. We generated for each graph and each method 4 landmarks. The heuristics have been run three times and a median of the runs has been taken. (*) indicates that no binary solution has been found in time.

Since we are not able to run the greedy algorithm on real-world road-networks, we limited ourself to compare to solution of the heuristics used so far to the one found by the greedy algorithm and therefore get an idea of how good the heuristics are for bigger graphs than those solvable by the integer linear program.

The results given in Table 5.2 indicates the same order as the comparison on slightly smaller graphs. Once again the greedy algorithm clearly dominates the other methods considering the resulting search space. Compared to the search space of the maxCover heuristic greedy as about 96 to 90 percent of the search space with the results indicating that the gap is getting bigger with larger graphs. Compared to the search space of the avoid heuristic the gap is with 94 to 88 percent of the search space a little bigger.

On the other hand, it is also obvious that if considering the time needed to compute the solution the greedy can not compare with the other methods even on small graphs. For the small graphs under 100 vertices as compared in Table 5.1, the greedy algorithm is around 1000 times slower than the heuristics, and for bigger graphs around 500-1500 vertices it even is around one million times slower than the heuristics. As the asymptotic complexity implies, the difference gets worse the bigger the graphs get.

So considering that we can only improve the overall search space by around 4 to 12 percent, the additionally needed amount of time is not paying off very well. In this setting, it would be interesting to know how the heuristic proposed in Section 4.4 performs compared to the so far leading heuristics like avoid, advanced avoid or maxCover.

GRAPHS	DEU		LUX	
	TIME	SEARCH SPACE	TIME	SEARCH SPACE
500 vert.			(370 vert.)	
avoid	0.0064 s	8,798,842	0.0049 s	3,767,726
adv. avoid	0.0091 s	9,115,170	0.0099 s	3,747,703
maxCover	0.0333 s	8,532,156	0.0254 s	3,561,492
greedy	1h 4m 6s	8,264,125	16m 19s	3,386,954
1000 vert.				
avoid	0.0126 s	60,047,064	0.0127 s	69,765,480
adv. avoid	0.0181 s	62,470,691	0.0182 s	69,831,648
maxCover	0.0667 s	59,082,280	0.0709 s	66,182,107
greedy	19h 40m 57s	56,708,679	15h 56m 34s	61,741,536
1500 vert.				
avoid	0.0189 s	171,144,678	0.0193 s	200,782,618
adv. avoid	0.0267 s	173,950,651	0.0276 s	209,381,550
maxCover	0.1096 s	172,310,899	0.1022 s	191,656,773
greedy	3d 17h 16m 48s	156,467,985	3d 14h 1m 19s	174,687,874
2000 vert.				
avoid	0.0252 s	679,638,147	0.0266 s	410,382,530
adv. avoid	0.0361 s	677,312,272	0.0375 s	437,572,399
maxCover	0.1368 s	652,973,350	0.1402 s	401,492,530
greedy	-	-	-	-

Table 5.2: Comparison of the heuristics used so far with the greedy algorithm on two different graphs. During each run we generated 16 landmarks. The heuristics have been run three times and the median of the runs has been taken.

6. Conclusion

In this thesis we studied the preprocessing phase of the ALT algorithm. In particular, we studied the effects landmarks have on the ALT search space to get a better understanding and to improve the selection of landmarks. Though the effects of landmarks on the ALT search space on a general graph are quite complex, we were able to derive several results for the effects on easier graph classes like chains and trees.
Since we have been able to show that a new, less complex search space model for ALT is appropriate we were able to reduce the problem of the selection of k landmarks to the maximum coverage problem. We are also able to formulate the greedy algorithm for the selection of k landmarks as a maximum coverage problem, for which we could apply approximation as well as inapproximability results. The formulation of the problem as a maximum coverage problem also results in an integer linear program we formulated and implemented to solve the problem optimally for small graphs. We also gave an algorithm for a new heuristic, which unfortunately could not be tested in this thesis but promises good results at least on trees due to its construction.

Future work: ALT is not the preprocessing based algorithm which results in the fastest query times. In fact there are actually several algorithms resulting in faster query times. But in times where research is leading towards time-dependent routing, ALT, which has the advantage that it is not necessary to repeat the preprocessing everytime the length of an edge has changed (though the search space may not be as good as before), is remaining interesting. Hence it would be interesting to see how the search space changes for routing in time-dependent graphs as well as what can be done to select landmarks with an overall (i.e. for each possible variant of the graph) good performance.

Bibliography

Bauer, R., Columbus, T., Katz, B., Krug, M., and Wagner, D. (2010). Preprocessing speed-up techniques is hard.

Bauer, R. and Delling, D. (2009). Sharc: Fast and robust unidirectional routing. *ACM Journal of Experimental Algorithmics*, 14.

Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press.

Delling, D., Sanders, P., Schultes, D., and Wagner, D. (2006). Highway hierarchies star. In *9th DIMACS Implementation Challenge*.

Delling, D., Sanders, P., Schultes, D., and Wagner, D. (2009). Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, pages 117–139.

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

Feige, U. (1998). A threshold of ln n for approximating set cover. *JOURNAL OF THE ACM*, 45:314–318.

Ford, L. and Fulkerson, D. (1962). *Flows in Networks*. Rand.

Geisberger, R., Sanders, P., Schultes, D., and Delling, D. (2008). Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, pages 319–333.

Goldberg, A. and Werneck, R. (2005). Computing point-to-point shortest paths from external memory. In *Proceedings Workshop on Algorithm Engineering and Experiments (ALENEX 2005)*. SIAM.

Goldberg, A. V. and Harrelson, C. (2004). Computing the shortest path: A* search meets graph theory. Technical report.

Goldberg, A. V., Kaplan, H., and Werneck, R. F. F. (2007). Better landmarks within reach. In *WEA*, pages 38–51.

Gutman, R. (2004). Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *Proceedings 6th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 100–111. SIAM.

Hochbaum, D. S. H., editor (1997). *Approximation algorithms for NP-hard problems*. PWS Publ. Comp., Boston [u.a.].

Holzer, M., Schulz, F., and Wagner, D. (2009). Engineering multilevel overlay graphs for shortest-path queries. *J. Exp. Algorithmics*, 13:2.5–2.26.

ILOG, I. (2009). Cplex optimizer. online, `http://www.ilog.com/products/cplex/`.

Schultes, D. and Sanders, P. (2007). Dynamic highway-node routing. In *WEA*, pages 66–79.

Wagner, D. and Willhalm, T. (2007). Speed-up techniques for shortest-path computations. In *STACS*, pages 23–36.