Energy-efficient Paths in Radio Networks*

Rene Beier[1], Stefan Funke[1], Domagoj Matijević[1], and Peter Sanders[2]

[1] Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany
[2] Universität Karlsruhe, Fakultät für Informatik, 76128 Karlsruhe, Germany

Abstract. We consider a radio network consisting of n stations represented as the complete graph on a set of n points in the Euclidean plane with edge weights $\omega(p,q) = |pq|^\delta + C_p$, for some constant $\delta > 1$ and nonnegative offset costs C_p. Our goal is to find paths of minimal energy cost between any pair of points that do not use more than some given number k of hops.
We present an exact algorithm for the important case when $\delta = 2$, which requires $\mathcal{O}(kn \log n)$ time per query pair (p,q). For the case of an unrestricted number of hops we describe a family of algorithms with query time $\mathcal{O}(n^{1+\alpha})$, where $\alpha > 0$ can be chosen arbitrarily. If we relax the exactness requirement, we can find an approximate $(1+\epsilon)$ solution in constant time by querying a data structure which has linear size and which can be build in $\mathcal{O}(n \log n)$ time. The dependence on ϵ is polynomial in $1/\epsilon$.
One tool we employ might be of independent interest: For any pair of points $(p,q) \in (P \times P)$ we can report in constant time the cluster pair (A,B) representing (p,q) in a well-separated pair decomposition of P.

1 Introduction

The *shortest-path* problem is one of the fundamental problems which has been studied for a long time. In this work, the following variant is examined: Given a set P of n points in $\mathbb{R}^2$ we consider the complete directed graph $G = (P, P \times P)$ with edge weights

$$\omega(p,q) = |pq|^\delta + C_p, \tag{1}$$

where $\delta > 1$ is a constant, C_p is a node-dependent offset cost and $|pq|$ denotes the Euclidean distance between p and q. For every path π, let $\omega(\pi)$ denote the length of π with respect to the length function. We say that a path $\pi(p,q)$ from p to q is a t-approximate shortest path if and only if $\omega(\pi(p,q)) \leq \omega(\pi'(p,q)) \cdot t$, for any other path $\pi'(p,q)$ from p to q. For every integer $k \geq 1$, a k-hop shortest p-q-path is a shortest path from p to q that uses at most k edges.

Motivation

The original motivation for our problem stems from applications in wireless networking. In recent years wireless network technology has gained tremendous importance. It not only opens new dimensions in the availability of high-bandwidth connections in particular for mobile applications, but also more and more replaces so far wired network installations. While the spatial aspect was already of interest in the wired network world due to cable costs etc., it has far more influence on the design and operation of wireless networks. The power required to transmit information via radio waves is heavily dependent on the Euclidean distance between sender and receivers and in fact depends *superlinearly* on this distance. Unfortunately, when measuring distances between radio stations in terms of the power required to communicate, we lose the nice properties of metric spaces where the triangle inequality holds, and many problems that are simple in a metric space become quite challenging.

* Preliminary versions of parts of the results described here were already published in the proceeding of 11th Annual European Symposium on Algorithms (see [FMS03]) and the proceeding of 29th International Colloquium on Automata, Languages and Programming (see [BSS02]).

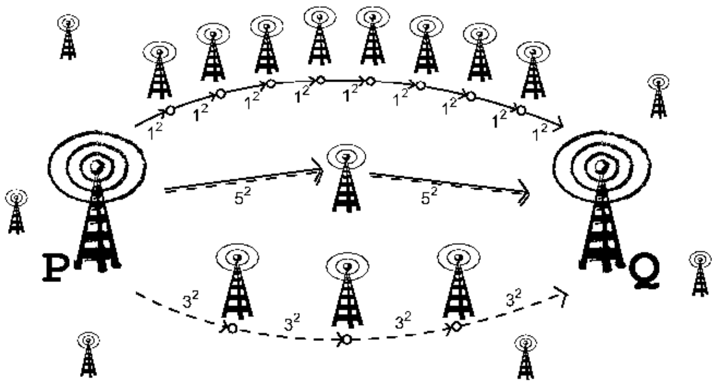

Fig. 1. *A radio network with* $9, 4, 2$*-hop paths from* P *to* Q *with costs 9, 36, 50.*

Picture the following scenario: during a long summer drought, wild fires have started in a large region of a remote nature preserve that is hardly accessible by ground transportation. To be able to continuously assess the situation and plan appropriate countermeasures, airplanes are sent out to deploy thousands of sensor nodes which are to gather, aggregate and transfer data to mission control. Naturally the lack of any established communication infrastructure within in such a nature preserve makes wireless communication the only possible way to interconnect the sensor nodes and enable them to report their sensed data. Apart from their sensing devices, each sensor node consists of a radio unit for transmitting/receiving messages via radio waves and a power supply (e.g. battery or solar panel). Communication takes place by a station transmitting a signal of a certain strength. Any other station being sufficiently close – as modelled by Equation (1) – can receive this message. The exponent δ is typically between 2 and 6; for $\delta = 2$ the edge weights reflect the exact energy requirement for free space communication. For larger values of δ we get a popular heuristic model for absorption effects [Rap96,Pat00]. The *offset cost* C_p accounts for the distance independent energy consumption of the wireless stations (e.g. the energy consumption of signal processing during sending and receiving). Communication between two stations that are not within their respective ranges can be achieved by multi-hop transmission, where intermediate stations serve as relays to forward the message. Even when two stations are within communication range, they still might prefer to use a multi-hop transmission in order to minimize the overall energy needed (see Figure 1). In fact, in our idealistic model we are going to assume that every station is within communication range of all others and will be mainly concerned with using the least possible energy needed to transmit messages between two different stations. Solely optimizing for energy consumption typically leads to multi-hop paths with many relay stations, which increases both latency as well as the probability of one of the many relay stations failing. Limiting the number of retransmissions to some fixed number k is a natural way to avoid such issues.

Related Work

The first attempt, to our knowledge, to cope with Dijkstra's quadratic complexity of the exact shortest path problem in a complete geometric graph with a superlinear distance function like $\omega(p, q) = f(|pq|^\delta)$ is by Chan, Efrat, and Har-Peled [EH98,CE01], modelling the fuel consumption of airplanes routed between a set P of airports. They observe that for $\omega(p, q)$ and $\delta \geq 2$, exact geometric shortest paths are equivalent to shortest paths in the Delaunay triangulation of P, i.e., optimal paths can be computed in time $\mathcal{O}(n \log n)$. This is rather easy to see since a non-Delaunay edge e on a shortest path contains points inside the smallest circle enclosing the edge (replacing e by two edges going via such a point will yield a smaller cost for $\delta \geq 2$). Note that this approach completely collapses for k-hop paths because most Delaunay edges are very short. The main contribution of Chan et al. is a sophisticated $\mathcal{O}(n^{4/3+\gamma})$ time algorithm for computing exact geometric shortest paths for monotone cost functions $\omega(p,q) = f(|pq|)$ where γ is any positive constant.

Not insisting on the *exact* shortest path/distance computation has allowed for drastic improvement in the running time. For example, Goel et al. [GIV01] used the idea of reducing the geometric shortest path problem to a small number of calls to a data structure for $(1+\epsilon)$-approximate nearest neighbors. For general undirected graphs and for any integer $r \geq 1$, Thorup and Zwick [TZ01] show how to construct a distance oracle which answers $(2r-1)$-approximate queries in $\mathcal{O}(r)$ time using a data structure that takes $\mathcal{O}(rmn^{1/r})$ (expected) construction time and essentially optimal $\mathcal{O}(rn^{1+1/r})$ space. They also show that the $(2r-1)$-approximate paths can be produced in constant time per edge. Since the query time is essentially bounded by a constant (if r is constant), Thorup and Zwick also refer to their queries as approximate distance oracles. Their algorithms are also simple and easy to implement efficiently.

Our results

We consider the following problem: Given a set P of n points in $\mathbb{R}^2$ and some constant k, report for a given query pair of points $p, q \in P$, a path $\pi = \pi(p,q) = v_0 v_1 v_2 \ldots v_l$, with vertices $v_i \in P$ and $v_0 = p, v_l = q$ which consists of at most k segments, i.e. $l \leq k$, such that its weight $\omega(\pi) = \sum_{0 \leq i < l} \omega(v_i, v_{i+1})$ is minimized. By $\pi_{opt} = \pi_{opt}(p,q)$ we denote an optimal path from p to q under this criterion.

In Section 2 we present exact algorithms for the above problem. We start with a very simple algorithm for the special case $\delta = 2$ based on Voronoi diagrams that finds optimal 2-hop paths for uniform offset costs in time $\mathcal{O}(\log n)$ after a preprocessing time of $\mathcal{O}(n \log n)$ and space $\mathcal{O}(n)$. Using dynamic programming this can easily be extended to an $\mathcal{O}(n \log n)$ algorithm for optimal paths with at most four hops. We then give a very general algorithm for finding optimal routes with up to k hops and arbitrary weight function which essentially has $\mathcal{O}(kn^2)$ running time for complete graphs. We show that in the special case $\delta = 2$ such a naive algorithm can be speeded up by reducing it to a special case of the 3D nearest neighbor problem which can be solved efficiently. The improved algorithm has running time $\mathcal{O}(kn \log n)$ and uses $\mathcal{O}(n)$ space. All the above algorithms are quite practical.

In Section 3 we construct approximate algorithms for the k-hop shortest path in radio networks. Our main result is a data structure that uses linear space and can be built in time $\mathcal{O}(n \log n)$ for any constants k, $\delta > 1$, and $\epsilon > 0$. In constant time it allows to compute k-hop paths between arbitrary query points that are within a factor $(1+\epsilon)$ from optimal. When k, δ, and ϵ are considered variables, the query time remains constant and the preprocessing time is bounded by a polynomial in k, δ, and $1/\epsilon$.

The algorithm has two main ingredients that are of independent interest. The first part, discussed in Section 3.2, is based on the observation that for approximately optimal paths it suffices to compute a shortest path using a constant-size subset of the points — one point for each square cell in some grid that depends on the query points. This subset can be computed in time $\mathcal{O}(\log n)$ using well known data structures supporting (approximate) quadratic range queries [BKOS,AM00]. These data structures and in particular their space requirement are independent of k, δ, and ϵ. Some variants even allow insertion and deletion of points in $\mathcal{O}(\log n)$ time. Section 3.3 discusses the second ingredient. Well separated pair decompositions [CK92] allow us to answer arbitrary approximate path queries by precomputing a linear number of queries. We develop a way to access these precomputed paths in constant time using hashing. This technique is independent of path queries and can be used for retrieving any kind of information stored in well separated pair decompositions. Since the query time is bounded by a constant, our result can be seen as an approximate *path oracle* for the radio networks.

Finally Section 4 gives a more theoretical solution for finding an exact shortest path for the case of $\delta = 2$ and an unlimited number of hops with running time and space $\mathcal{O}(n^{1+\epsilon})$ for any constant $\epsilon > 0$. The solution is based on 3D closest bichromatic pair queries [Epp95] similar to the general algorithm in [CE01]. We mainly improve the nearest neighbor queries involved to take advantage of the special structure of the problem.

All algorithms we give here are sequential centralized algorithms. Although in wireless networks, distributed algorithms are in principle more desirable, we believe that centralized algorithms are a necessary starting point. In particular, it seems that many distributed algorithms conceivable for the stated problems would be more expensive than just maintaining the network description at selected (possibly redundant) server nodes which handle routing requests by the other nodes. Query costs can often be amortized over many messages exchanged along the computed route. Such a client-server setting also takes into account that many wireless networks have some nodes that have higher computational capabilities and more ample power (e.g. in cars or buildings). Even if at the end fully distributed algorithms are preferred, centralized algorithms are a good reference point to analyze the tradeoff between computation cost and quality of the solutions.

2 Computing Optimal k-hop Paths

In the following we will begin with a simple algorithm for computing optimal k-hop shortest paths which works for arbitrary connected graphs (not necessary geometric ones) and an arbitrary weight function. Perhaps more interestingly, for $\delta = 2$, the problem has a special geometric structure that we can exploit for efficient solutions.

2.1 Dynamic programming approach

In this section we describe a dynamic programming algorithm for computing k-hop shortest path. It can be seen as a variant of the Bellmann/Ford algorithm[$\star\star\star$] that executes only k instead of $n-1$ iterations and exploits geometry in order to speed up the relaxation of edges in each iteration.

For any $s \in P$, let $\mu_k(s)$ denote the cost of a k-hop shortest path from the source node p to s. Clearly, $\mu_1(s) = C_p + |ps|^\delta$, for all $s \in P\backslash\{p\}$ and $\mu_0(p) = 0$. For $i \geq 2$ the Bellmann/Ford recurrence yields

$$\mu_i(s) = \min\{\mu_{i-1}(s), \min_{r \in P}\{\mu_{i-1}(r) + C_r + |ps|^\delta\}\}. \tag{2}$$

A naive implementation yields an algorithm with running time $\Theta(kn^2)$: We start by computing the function μ_1 and iteratively determine μ_i for all $i \in \{2, \ldots, k\}$ by applying equation 2. For the important case of $\delta = 2$ we can exploit the geometric structure of the problem, and reduce the running time of each iteration from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$.

Theorem 1. *In a radio network with edge weights $C_p + |pq|^2$, the single source shortest path problem restricted to paths with at most k-hops can be solved in time $\mathcal{O}(kn \log n)$ using space $\mathcal{O}(n)$.*

Proof. The problem of finding $r \in P$ that minimizes $\mu_{i-1}(r) + C_r + |rs|^2$ on the right hand side of equation (2) can be reduced to a 3D nearest neighbor query. The idea is to embed the point set P into R^3 and use the additional dimension to encode the cost of a shortest i-hop path from p to r as well as the offset cost C_r. Define, for $1 \leq i \leq k$, $f_i : P \rightarrow \mathbb{R}^3; f_i(r) = (x_r, y_r, \sqrt{\mu_i(r) + C_r})$, where $r = (x_r, y_r)$. The embedding of the original "unlifted" points are given by $f_0 : P \rightarrow \mathbb{R}^3; f_0(r) = (x_r, y_r, 0)$. For each pair of points (r, s) we see

$$|f_i(r)f_0(s)|^2 = |rs|^2 + \left(\sqrt{\mu_i(r) + C_r}\right)^2 = \mu_i(r) + C_r + |rs|^2\,. \tag{3}$$

Hence, for any $s \in P$, finding $r \in P$ that minimizes $\mu_i(r) + C_r + |rs|^2$ is equivalent to finding the closest point to $f_0(s)$ among all points in $\{f_i(r) \mid r \in P, r \neq s\}$.

[$\star\star\star$] For the original method of what is nowadays known as Bellman/Ford algorithm see Bellmann [Bel57] and Ford [For56].

Notice, that equation (3) corresponds to the power function of a 2D-power diagram for sites P, where site $r \in P$ has weight $-\sqrt{\mu_i(r) + C_r}$. Therefore, nearest neighbor queries can be answered in time $\mathcal{O}(\log n)$ after $\mathcal{O}(n \log n)$ preprocessing [Aur87]. Each phase takes $\mathcal{O}(n \log n)$ time for building the power diagram and $\mathcal{O}(n \log n)$ time for n nearest neighbor queries. Hence, the overall running time is $\mathcal{O}(kn \log n)$. □

For a practical implementation of the above algorithm, one can exploit the fact that all n nearest neighbor queries in one iteration are independent of each other. Hence, it suffices to implement a *batched* nearest neighbor algorithm that runs in time $\mathcal{O}(n \log n)$. We expect that this is a significant constant factor faster than answering n subsequent nearest neighbor queries that have to traverse some complicated nearest neighbor data structure. For example, one can build a Power diagram without the point location data structure and then perform the batched point location using a simple sweep line algorithm. One can even design a sweep line algorithm for batched nearest neighbor queries without constructing a Power diagram. This might further speed up the search and decreases the memory requirements of the algorithm.

2.2 Two-hop queries in logarithmic time

By using the same idea of encoding additional costs for each relay station by lifting up the points in the third dimension, we can answer queries for the shortest two-hop path in logarithmic time.

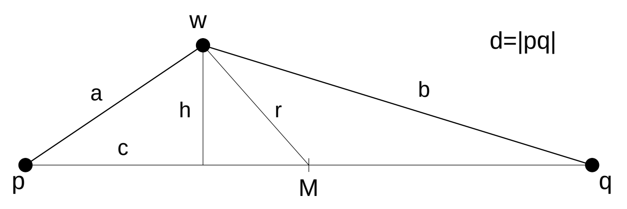

Fig. 2. 2-hop path between source p and target q. Illustration of the proof of Theorem 2.

Theorem 2. *Given an arbitrary source and destination p and q in a radio network with edge weights $C_p + |pq|^2$, we can find a radio station w minimizing $d(p,w) + d(w,q)$ in time $\mathcal{O}(\log n)$ after preprocessing time $\mathcal{O}(n \log n)$ using $\mathcal{O}(n)$ space.*

Proof. Given source p and target q, we can express the cost $c(w)$ of the 2-hop path (p,w,q) as a function of $d = |pq|$, C_w and r, the distance between the point w and the midpoint M of the line segment pq (see Figure 2):

$$\begin{aligned} h^2 &= r^2 - (d/2 - c)^2 = r^2 - d^2/4 - c^2 + dc \\ a^2 + b^2 &= c^2 + h^2 + (d-c)^2 + h^2 = d^2 + 2c^2 - 2dc + 2h^2 = d^2/2 + 2r^2 \\ c(w) &= C_p + a^2 + C_w + b^2 = C_p + d^2/2 + 2(r^2 + C_w/2) \end{aligned}$$

Hence, we need to pick the relay station w that minimizes $r^2 + C_w/2$. This is equivalent to finding the point from $\{(x_w, y_w, \sqrt{C_w/2}) \mid w \in P\}$ which is closest to the midpoint M. Again, we can use a power diagram and a point location data structure to find such a w in logarithmic time. □

3 Approximate k-hop Path Queries

In this section we are concerned with computation of a k-hop shortest path that is "almost" optimal. Namely, for a given pair of query points p, q and an arbitrarily small constant $\epsilon > 0$ our primary objective will be to output a path $\pi(p,q)$ such that its weight is at most $(1+\epsilon) \cdot \omega(\pi_{opt})$.

In the following we assume that the weight function ω is of the form $\omega(p,q) = |pq|^\delta + C_p$ with $\delta > 1$ (the case $\delta \leq 1$ is trivial as we just need to connect p and q directly by one hop).

3.1 Preliminaries

Before we introduce our procedure for reporting approximate k-hop paths, we need to refer to some standard data structures from Computational Geometry which will be used in our algorithm.

Theorem 3 (Exact Range Query). *Given a set P of n points in $\mathbb{R}^2$ one can build a data structure of size $\mathcal{O}(n \log n)$ in time $\mathcal{O}(n \log n)$ which for a given axis aligned query rectangle $R = [x_l, x_u] \times [y_l, y_u]$ reports in $\mathcal{O}(\log n)$ time that R either contains no point or outputs a point $p \in P \cap R$.*

The data structure can be maintained dynamically such that points can be inserted and deleted in $\mathcal{O}(\log n \log\log n)$ amortized time. The preprocessing time then increases to $\mathcal{O}(n \log n \log\log n)$ and the query time to $\mathcal{O}(\log n \log\log n)$. All the $\log\log n$ factors can be removed if only either *insertions* or *deletions are allowed.*

Proof. We use the standard 2-level range-tree construction. From the resulting $\mathcal{O}(\log^2 n)$ query time we can get rid of one $\log n$ by fractional cascading, see [BKOS]. The whole construction can be maintained dynamically allowing insertions and deletions using the results in [MN90]. $\square$

In fact, the algorithm we will present will also work with an approximate range reporting data structure such as the one presented in [AM00,AM98]. The part of their result relevant for us can be stated in the following theorem:

Theorem 4 (Approximate Range Query). *Given a set P of n points in $\mathbb{R}^2$ one can build a data structure of size $\mathcal{O}(n)$ in time $\mathcal{O}(n \log n)$ which for a given axis aligned query rectangle $R = [x_l, x_u] \times [y_l, y_u]$ with diagonal ω and any $\alpha > 0$ reports in $\mathcal{O}(\log n + \frac{1}{\alpha})$ time that the rectangle $R' = [x_l + \alpha\omega, x_u + \alpha\omega] \times [y_l + \alpha\omega, y_u + \alpha\omega]$ either contains no point or outputs a point $p \in P \cap R'$.*

The data structure can be maintained dynamically such that points can be inserted and deleted in $\mathcal{O}(\log n)$ time.

Basically this approximate range searching data structure works well if the query rectangle is fat; and since our algorithm we present in the next section will only query square rectangular regions, all the results in [AM00] and [AM98] apply. In fact we do not even need α to be very small, $\alpha = 1$ turns out to be good enough as we will see later on. So the use of an approximate range searching data structure helps us to get rid of the $\log n$ factor in space and some $\log\log n$ factors for the dynamic version. But to keep presentation simple we will assume for the rest of this section that we have an exact range searching data structure at hand. Furthermore, let us briefly cite two well known inequalities that turn out to be of use in the incoming analysis.
Minkowski's inequality For every $\delta > 1$ and a_i, $b_i > 0$, Minkowski's sum inequality states that

$$\left(\sum_{i=1}^{k}(a_i + b_i)^\delta\right)^{\frac{1}{\delta}} \leq \left(\sum_{i=1}^{k} a_i^\delta\right)^{\frac{1}{\delta}} + \left(\sum_{i=1}^{k} b_i^\delta\right)^{\frac{1}{\delta}}.$$

Equality holds iff the sequences $a_1, a_2, \ldots$ and $b_1, b_2, \ldots$ are proportional.
Hölder's inequality Let

$$\frac{1}{p} + \frac{1}{q} = 1$$

with p, $q > 1$. Then, for every a_i, $b_i > 0$ Hölder's inequality for sums states that

$$\sum_{i=1}^{k} a_i b_i \leq \left(\sum_{i=1}^{k} a_i^p\right)^{\frac{1}{p}} \left(\sum_{i=1}^{k} b_i^q\right)^{\frac{1}{q}}.$$

Equality holds when $b_i = c a_i^{p-1}$, for some $c > 0$.

3.2 Computing Approximate k-hop Paths

We will now focus on how to process a k-hop path query for a pair of points p and q assuming that we have already constructed the data structure for orthogonal range queries (which can be done in time $\mathcal{O}(n \log n)$).

Lemma 1. *For the optimal path π_{opt} connecting p and q we have $\frac{|pq|^\delta}{k^{\delta-1}} + C_p \leq |\pi_{\text{opt}}| \leq |pq|^\delta + C_p$.*

Proof. As we can connect p and q directly using one hop, the upper bound follows immediately. For the lower bound observe that the cheapest way to connect p and q is to divide the segment pq into k subsegments of equal length, which yields the lower bound. □

Definition 1. *We define the axis-aligned square of side-length l centered at the midpoint of a segment pq as the* frame *of p and q, $F(pq, l)$.*

Lemma 2. *The optimal path π_{opt} connecting p and q lies within the frame $F(pq, k^{(\delta-1)/\delta}|pq|)$ of p and q.*

Proof. Assume that the optimal path visits some point r outside the frame F. Note that such a path has Euclidean length at least $|pr| + |rq| > k^{(\delta-1)/\delta}|pq|$ and therefore the cost is lower bounded by $\frac{(|pr|+|rq|)^\delta}{k^{\delta-1}} + C_p > \frac{(k^{(\delta-1)/\delta}|pq|)^\delta}{k^{\delta-1}} + C_p = |pq|^\delta + C_p$ which in turn implies that doing one direct hop from p to q is even better. □

We are now armed to state our algorithm to compute a k-hop path which, as we will later show, is a $(1+\epsilon)$ approximation to the optimal k-hop path from p to q.

```
k-hop-Query($\boldsymbol{p, q, \epsilon}$)
1. Put a grid of cell-width $\alpha \cdot |pq|/k$ on the frame F$(pq, k^{(\delta-1)/\delta}|pq|)$ with $\alpha = \frac{\ln(2)}{2\sqrt{2}} \cdot \frac{\epsilon}{\delta}$.
2. For each grid cell $C$ perform an orthogonal range query to either certify that the cell is empty
   or report a point with smallest offset cost[†] which will serve as a representative for $C$.
3. Compute the optimal $k$-hop path $\pi(p, q)$ with respect to all representatives and $\{p, q\}$.
4. Return $\pi(p, q)$
```

Please look at Figure 3 for a schematic drawing of how the algorithm computes the approximate k-hop path. It remains to argue about correctness and running time of our algorithm. Let us first consider its running time.

Lemma 3. k-hop-Query$(p,\ q,\ \epsilon)$ *can be implemented to return a result in time $\mathcal{O}(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\epsilon^2} \cdot T_R(n) + T_{k,\delta}(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\epsilon^2}))$, where $T_R(n)$ denotes the time for one 2-dimensional range query on the original set of n points and $T_{k,\delta}(x)$ denotes the time for the exact computation of a minimal k-hop path for one pair amongst x points under the weight function $\omega(pq) = |pq|^\delta + C_p$.*

Proof. By the choice of frame and cell width, it is easy to see that in the first step of our algorithm we generate a grid of $\mathcal{O}((\frac{\delta \cdot k^{(2\delta-1)/\delta}}{\epsilon})^2)$ many different cells which is $\mathcal{O}(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\epsilon^2})$. For each of the cells we perform an orthogonal range query each of which takes $T_R(n)$ time. For all the representatives which we have found, we run an exact minimum k-hop path algorithm which takes $T_{k,\delta}(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\epsilon^2})$. □

Let us now turn to the correctness of our algorithm, i.e. for any given positive ϵ, we want to show that our algorithm returns a k-hop path of weight at most $(1+\epsilon)$ times the weight of the optimal path. We will show that using only representatives of grid cells there exists a path of at most this weight. In the following we assume that the optimal path π_{opt} consists of a sequence of points $p_0 p_1 \ldots p_j$, $j \leq k$ and $l_i = |p_{i-1}p_i|$.

Before we get to the actual proof of this claim, we need to state a small technical lemma.

[†] This can be easily incorporated into the standard geometric range query data structures.

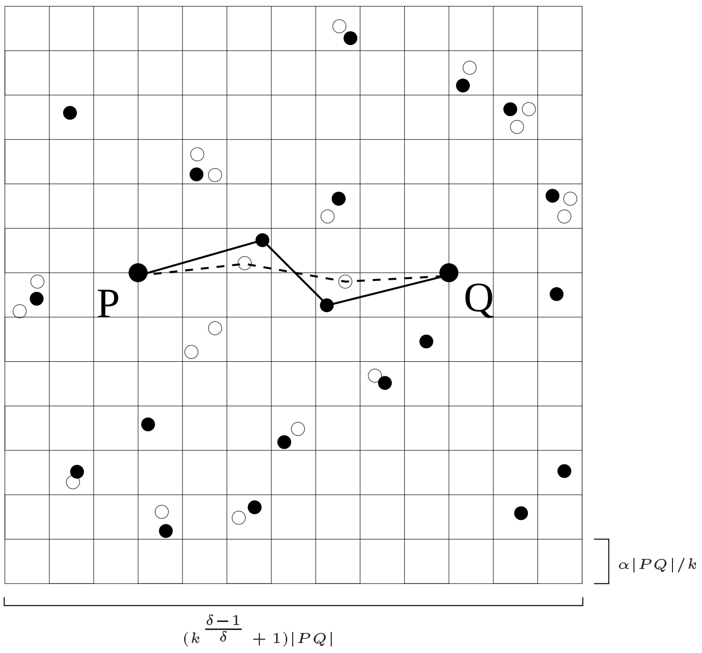

Fig. 3. 3-*hop-query for P and Q: representatives for each cell are denoted as solid points, the optimal path is drawn dotted, the path computed by the algorithm is drawn solid*

Lemma 4. *For any $\delta > 1$ and $l_i, \xi > 0$ the following inequality holds*

$$\frac{\sum_{i=1}^{k}(l_i+\xi)^\delta}{\sum_{i=1}^{k} l_i^\delta} \le \left(\frac{\sum_{i=1}^{k}(l_i+\xi)}{\sum_{i=1}^{k} l_i}\right)^\delta$$

Proof. Note that Minkowski's inequality directly implies

$$\left(\sum_{i=1}^{k}(l_i+\xi)^\delta\right)^{\frac{1}{\delta}} \le \left(\sum_{i=1}^{k} l_i^\delta\right)^{\frac{1}{\delta}} + \left(\sum_{i=1}^{k}\xi^\delta\right)^{\frac{1}{\delta}} = \left(\sum_{i=1}^{k} l_i^\delta\right)^{\frac{1}{\delta}} + k^{\frac{1}{\delta}}\cdot\xi$$

which in turn gives

$$\left(\frac{\sum_{i=1}^{k}(l_i+\xi)^\delta}{\sum_{i=1}^{k} l_i^\delta}\right)^{\frac{1}{\delta}} \le 1 + \frac{k^{\frac{1}{\delta}}\cdot\xi}{(\sum_{i=1}^{k} l_i^\delta)^{\frac{1}{\delta}}}.$$

Moreover, note that Hölder's inequality implies

$$\left(\frac{\sum_{i=1}^{k} l_i^\delta}{k}\right)^{\frac{1}{\delta}} \ge \frac{\sum_{i=1}^{k} l_i}{k}$$

for $b_i = 1$, $i = 1, \ldots, k$, $p = \delta$ and $q = \frac{\delta}{\delta-1}$, which completes the proof. □

Lemma 5. K-HOP-QUERY$(p,\ q,\ \epsilon)$ *computes a k-hop path from p to q of weight at most $(1+\epsilon)\omega(\pi_{\text{opt}}(p,q))$ for $0 < \epsilon \le 1$.*

Proof. In order to prove the claim, we will compare the actual optimum path π_{opt} with the path using the representatives of the grid cells containing the points of π_{opt}. Note, however, that this is not necessarily the path returned by the algorithm since it might find a better path using representatives of other cells.

Let ξ denote the possible error incurred by taking the respective representative edge. Using Lemma 4, it is sufficient to show that

$$\left(\frac{\sum_{i=1}^{k}(l_i+\xi)}{\sum_{i=1}^{k} l_i}\right)^{\delta} \leq 1+\epsilon.$$

Knowing that the absolute "detour"[‡] incurred by replacing its endpoints by the respective representatives in their grid cell is bounded by $\xi \leq 2\sqrt{2}\alpha\frac{d}{k}$ we have

$$\left(1+\frac{k\cdot 2\sqrt{2}\alpha\frac{d}{k}}{\sum_{i=1}^{k} l_i}\right)^{\delta} \leq 1+\epsilon.$$

Thus, observing that $(\sum_{i=1}^{k} l_i)/d$ is at least 1, we get the following bound on α

$$\alpha \leq \frac{(1+\epsilon)^{1/\delta}-1}{2\sqrt{2}}.$$

For any positive α less then the above upper bound claim of the lemma holds. However, we would like to have α as large as possible in a given bound but also a little bit more nicely formulated. Hence, using the well known fact that $(1+1/x)^x \leq e$ for any $x>0$, we get

$$(1+\epsilon) = e^{\ln(1+\epsilon)} \geq \left(\left(1+\frac{\ln(1+\epsilon)}{\delta}\right)^{\frac{\delta}{\ln(1+\epsilon)}}\right)^{\ln(1+\epsilon)}$$

which easily reduces to

$$\delta\cdot\left((1+\epsilon)^{1/\delta}-1\right) \geq \ln(1+\epsilon).$$

Furthermore, $\ln(1+\epsilon) \geq \ln(2)\cdot\epsilon$ for any $0<\epsilon\leq 1$ and therefore it suffice to choose

$$\alpha = \frac{\ln(2)}{2\sqrt{2}}\cdot\frac{\epsilon}{\delta}.$$

□

Summary

In our approximation algorithm we reduced the problem of computing an approximate k-hop path from p to q to one exact k-hop path computation of a small, i.e. constant number of points (only depending on k,δ and ϵ). Using the naive dynamic programming algorithm from Section 2.1 for computing optimal k-hop shortest paths, we get the following theorem (we give the bound for the case where an approximate range query data structure as mentioned in Theorem 4 is used).

Theorem 5. *We can construct a dynamic data structure allowing insertions and deletions with $\mathcal{O}(n)$ space and $\mathcal{O}(n\log n)$ preprocessing time such that $(1+\epsilon)$ approximate minimum k-hop path queries in radio networks can be answered in time $\mathcal{O}(\frac{\delta^2\cdot k^{(4\delta-2)/\delta}}{\epsilon^2}\cdot\log n+\frac{\delta^4\cdot k^{(7\delta-2)/\delta}}{\epsilon^4})$.*

The query time does not change when using exact range query data structures, only space, preprocessing and update times get slightly worse (see Theorem 3).

In Theorem 1 we presented an algorithm which for the special case $\delta=2$ computes the optimal k-hop path in time $\mathcal{O}(kn\log n)$. Applied to our problem we get the following corollary:

Corollary 1. *The subroutine of our algorithm to solve the exact k-hop shortest path problem given the set of $\mathcal{O}(\frac{k^3}{\epsilon^2})$ representative points can be solved in time $\mathcal{O}(\frac{k^5}{\epsilon^2}\cdot\log\frac{k}{\epsilon})$ if $\delta=2$.*

[‡] Here the analysis has to be changed slightly when using approximate range reporting data structures: for $\alpha=1$ we might get twice the "detour" assumed here.

3.3 The Approximate Path Oracle

In the previous section we have seen how to answer a (p, q) query in $\mathcal{O}(\log n)$ time (considering k, δ, ϵ as constants). Standard range query data structures were the only precomputed data structures used. Now we explain how additional precomputation can further reduce the query time. We show how to precompute a linear number of k-hop paths using the well-separated pair decomposition (WSPD), such that for every (p, q) pair, a slight modification of one of these precomputed paths is a $(1+\epsilon)(1+2\psi)^2$ approximate k-hop path and such a path can be accessed in constant time. Here $\psi > 0$ is the error incurred by the use of these precomputed paths and can be chosen arbitrarily small (the size of the WSPD then grows, though).

The Well-Separated Pair Decomposition We will first briefly introduce the so-called *well-separated pair decomposition* due to Callahan and Kosaraju ([CK92]).

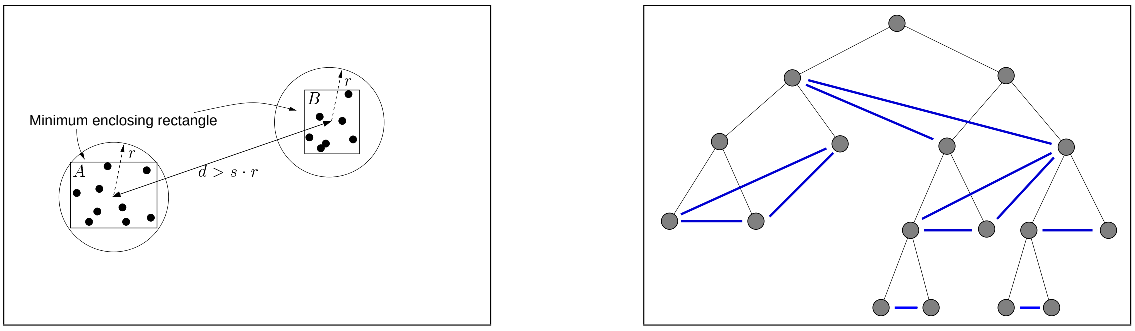

Fig. 4. Clusters A and B are "well-separated" if $d > s \cdot r$ (left). Example of split tree with additional blue edges (right).

The *split-tree* of a set P of points in $\mathbb{R}^2$ is the tree constructed by the following recursive algorithm:

```
SplitTree($P$)
1. if size($P$)=1 then return leaf($P$)
2. partition $P$ into sets $P_1$ and $P_2$ by halving its minimum enclosing axis-aligned rectangle $R(P)$
   along its longest dimension
3. return a node with children (SplitTree($P_1$), SplitTree($P_2$))
```

Although such a tree might have linear depth and therefore a naive construction as above takes quadratic time, Callahan and Kosaraju in [CK92] have shown how to construct such a binary tree in $\mathcal{O}(n \log n)$ time. With every node of that tree we can conceptually associate the set A of all points contained in its subtree as well as their minimum enclosing rectangle $R(A)$. By $r(A)$ we denote the radius of the minimum enclosing disk of $R(A)$, by center($R(A)$) the center of the minimum enclosing disk of $R(A)$. We will also use A to denote the node associated with the set A if we know that such a node exists.

For two sets A and B associated with two nodes of a split tree, $d(A, B)$ denotes the distance between the centers of the minimum enclosing balls of $R(A)$ and $R(B)$ respectively. A and B are said to be *well-separated* if $d(A, B) > s \cdot r$, where r denotes the radius of the larger of the two minimum enclosing balls of $R(A)$ and $R(B)$ respectively (see Fig. 4). s is called the *separation constant.*

In [CK92], Callahan and Kosaraju present an algorithm which, given a split tree of a point set P with $|P| = n$ and a separation constant s, computes in time $\mathcal{O}(n(s^2 + \log n))$ a set of $\mathcal{O}(n \cdot s^2)$ additional *blue* edges for the split tree, such that

– the point sets associated with the endpoints of a blue edge are well-separated with separation constant s.
– for any pair of leaves (a, b), there exists exactly one blue edge that connects two nodes on the paths from a and b to their lowest common ancestor $lca(a, b)$ in the split tree

The split tree together with its additional blue edges is called the *well-separated pair decomposition (WSPD).*

Using the WSPD for Precomputing Path Templates In fact the WSPD is exactly what we need to efficiently precompute k-hop paths for all possible $\Theta(n^2)$ path queries. So we will use the following preprocessing algorithm:

1. compute a well-separated pair decomposition of the point set with the separation constant $s = k^{(\delta-1)/\delta} \cdot 8\delta \cdot \frac{1}{\psi}$
2. for each *blue* edge compute a $(1+\epsilon)$-approximation to the lightest k-hop path between the centers of the associated bounding rectangles.

With the above algorithm, we have precomputed an approximate k-hop path between center points of each well-separated cluster pair. Note that center points of associated bounding rectangles are not necessarily in P. Without loss of generality, we assume that the center point a of some cluster A has the largest offset cost in A, say $C_a = C_A^{\max} := \max\{C_{a'}, a' \in P \cap A\}$.

At query time, for a given query pair (p, q), it remains to find the unique blue edge (A, B) which links a node of the path from p to $lca(p, q)$ to a node of the path from q to $lca(p, q)$. We take the precomputed k-hop path associated with this blue edge, replace its first and last node by p and q respectively and return this modified path (see Figure 5).

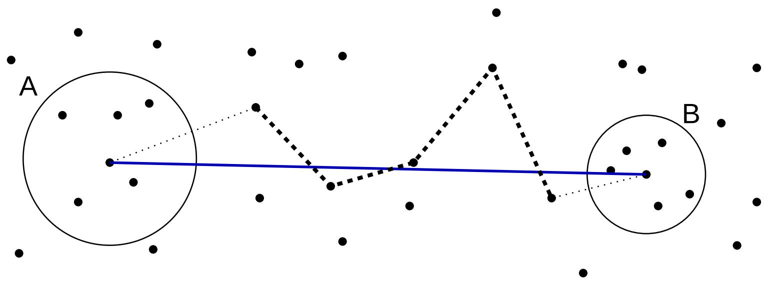

Fig. 5. *Example of the blue-edge connecting well-separated sets A and B and the template path (thick dashed line) saved with it.*

In the following we will show that the returned path is indeed a $(1+\epsilon)(1+2\psi)^2$ approximation of the lightest k-hop path from p to q. Later we will also show that this path can be found in constant time. For the remainder of this section let $\pi_{\text{opt}}^P(x, y)$ denote the optimal k-hop path between two points x, y not necessarily in P such that all hops have starting and end point in P (except for the first and last hop). We first start with a lemma which formalizes the intuition that the length of an optimal k-hop path does not change much when perturbing the query points slightly.

Lemma 6. *Given a set of points P and two pairs of points (a, b) and (a', b') with $d(a, b) = d$ and $d(a, a') \leq c$, $d(b, b') \leq c$ with $c \leq \frac{\psi d}{k^{(\delta-1)/\delta} \cdot 4\delta}$, then we have $\omega(\pi_{opt}^P(a', b')) \leq (1+2\psi)\omega(\pi_{opt}^P(a, b))$.*

Proof. Consider the path $\pi_{opt}^P(a, b) = v_0 v_1 \ldots v_k$ with $v_0 = a$, $v_k = b$ and consider its modification $\pi' = a' v_1 \ldots v_{k-1} b'$. For the analysis we will distinguish between the cases where the edge (a, v_1) ((v_{k-1}, b) respectively) is "long" or "short". If $(c + |av_1|)^\delta \leq (1+\psi)|av_1|^\delta$, we are safe and this is the case when $|av_1|$ is *long* edge, namely $|av_1| \geq \frac{c}{(1+\psi)^{1/\delta}-1}$. So let us now consider the case that (a, v_1) is short. Clearly an upper bound on the cost of $\pi^P_{opt}(a', b')$ is given by:

$$\begin{aligned}\omega(\pi^P_{opt}(a', b')) &\leq \omega(\pi') = \omega(\pi^P_{opt}(a, b)) - (|av_1|^\delta + C^{\max}_A) + |a'v_1|^\delta + C_{a'} - \\ &- (|bv_{k-1}|^\delta - C_{v_{k-1}}) + |b'v_{k-1}|^\delta + C_{v_{k-1}} \\ &\leq \omega(\pi^P_{opt}(a, b)) - |av_1|^\delta + (|av_1| + c)^\delta - |bv_{k-1}|^\delta + (|bv_{k-1}| + c)^\delta\end{aligned}$$

And since for $0 \leq x < y, \delta \geq 1$ we have $(c+x)^\delta - x^\delta \leq (c+y)^\delta - y^\delta$ and the respective edges are short, we can continue with

$$\begin{aligned}&\leq \omega(\pi^P_{opt}(a, b)) + 2 \cdot ((\frac{c}{(1+\psi)^{1/\delta}-1} + c)^\delta - (\frac{c}{(1+\psi)^{1/\delta}-1})^\delta) \\ &= \omega(\pi^P_{opt}(a, b)) + 2c^\delta \cdot ((\frac{(1+\psi)^{1/\delta}}{(1+\psi)^{1/\delta}-1})^\delta - (\frac{1}{(1+\psi)^{1/\delta}-1})^\delta) \\ &= \omega(\pi^P_{opt}(a, b)) + 2c^\delta \cdot \frac{\psi}{((1+\psi)^{1/\delta}-1)^\delta} \\ &\leq \omega(\pi^P_{opt}(a, b)) + 2c^\delta\psi \cdot (\frac{2\delta}{\psi})^\delta \\ &\leq \omega(\pi^P_{opt}(a, b))(1 + 2c^\delta\psi \cdot (\frac{2\delta}{\psi})^\delta \cdot \frac{k^{\delta-1}}{d^\delta})\end{aligned}$$

So finally it just remains to choose c such that $2 \cdot (\frac{2\delta c}{d})^\delta \cdot \frac{k^{\delta-1}}{\psi^{\delta-1}} \leq \psi$ which is certainly true for $c \leq \frac{\psi d}{k^{(\delta-1)/\delta} \cdot 4\delta}$ □

The following corollary of the above Lemma will be used later in the proof:

Corollary 2. *Given a set of points P and two pairs of points (a, b) and (a', b') with $d(a, b) = d$ and $d(a, a') \leq c$, $d(b, b') \leq c$ with $c \leq \frac{\psi d}{k^{(\delta-1)/\delta} \cdot 8\delta}$, then we have $\omega(\pi^P_{opt}(a', b')) \leq (1 + 2\psi)\omega(\pi^P_{opt}(a, b))$ as well as $\omega(\pi^P_{opt}(a, b)) \leq (1 + 2\psi)\omega(\pi^P_{opt}(a', b')$.*

Proof. Clearly the first claim holds according to Lemma 6. For the second one observe that $d' = |a'b'| \geq d - 2 \cdot c$ and then apply the Lemma again. □

Applying this Corollary, it is now straightforward to see that the approximation ratio of the modified template path is $(1 + 2\psi)^2(1 + \epsilon)$.

Lemma 7. *Given a well separated pair decomposition of a point set $P \subset \mathbb{R}^2$ with separation constant $s = \frac{k^{(\delta-1)/\delta} \cdot 8\delta}{\psi}$, the path $\pi(p, q)$ returned for a query pair (p, q) is a $(1 + 2\psi)^2(1 + \epsilon)$ approximate k-hop path from p to q.*

Proof. Let (A, B) be the unique cluster pair connected by a blue edge with $p \in A$, $q \in B$, c_A, c_B their respective cluster centers. By the choice of the separation parameter and Lemma 6, we know that $|pc_A|, |qc_B| \leq \frac{\psi |c_A c_B|}{k^{(\delta-1)/\delta} \cdot 8\delta}$ and therefore $\omega(\pi(p, q)) \leq (1 + 2\psi)(1 + \epsilon)\omega(\pi_{opt}(c_A, c_B)) \leq (1 + 2\psi)^2(1 + \epsilon)\omega(\pi(p, q))$. □

Adequate Choice for ψ and ϵ For a given $\phi > 0$, what is the right choice for ψ and ϵ to obtain an arbitrary approximation quality of $(1 + \phi)$? In order to answer on that, note that in the preprocessing phase for $\mathcal{O}(s^2 n)$ many different pairs of points we have to compute approximate paths in $\mathcal{O}(\frac{\log n}{\epsilon^2})$ time. Having in mind that separation constant $s = \mathcal{O}(\frac{1}{\psi})$, we have that preprocessing running time depends on ϵ and ψ as $\mathcal{O}(\frac{1}{\psi^2\epsilon^2} \cdot n \log n)$. Hence, we would like to choose ψ and ϵ such that $\frac{1}{\psi^2\epsilon^2}$ is minimized subject to the constraint $(1 + 2\psi)^2(1 + \epsilon) = (1 + \phi)$. This is the classical problem of finding constrained extreme values of the function with two real variables which can be solved using *Lagrange multiplier*.

We leave the details for the reader to figure out the right choice for ψ and ϵ to obtain an arbitrary approximation quality of $(1 + \phi)$.

Retrieving Cluster Pairs for query points in $\mathcal{O}(1)$ time In the previous paragraphs we have shown that using properties of the well-separated pair decomposition, it is possible to compute $\mathcal{O}(n)$ "template paths" such that for any query pair (p, q) out of the $\Omega(n^2)$ possible query pairs, there exists a good template path which we can modify to obtain a good approximation to the lightest k-hop path from p to q. Still, we have not shown yet how to determine this good template path for a given query pair (p, q) in constant time. We note that the following description does not use any special property of our original problem setting, so it may apply to other problems, where the well-separated pair decomposition can be used to encode in $\mathcal{O}(n)$ space sufficient information to cover a query space of $\Omega(n^2)$ size.

Gridding the Cluster Pairs The idea of our approach is to round the centers c_A, c_B of a cluster pair (A, B) which is part of the WSPD to canonical grid points $\widetilde{c_A}, \widetilde{c_B}$ such that for any query pair (p, q), $p \in A$ and $q \in B$, we can determine $\widetilde{c_A}, \widetilde{c_B}$ in constant time. The resolution of the grid used will depend on the distance of the query points. Furthermore we will show that there is only a constant number of cluster pairs (A', B') which have their cluster centers rounded to the same grid positions $\widetilde{c_A}, \widetilde{c_B}$, so well-known hashing techniques can be used to store and retrieve the respective blue edge and also some additional information $I_{(A,B)}$ (in our case: the precomputed k-hop path) associated with that edge.

In the following we assume that we have already constructed a WSPD of the point set P with a separation constant $s > 4$. For any point $p \in \mathbb{R}^2$, let snap(p, w) denote the closest grid-point of the grid with cell-width w originated at $(0, 0)$ and let $\mathcal{H} : \mathbb{Z} \times \mathbb{Z}^2 \times \mathbb{Z}^2 \rightarrow (I \times E)^*$ denote a hash table data structure which maps a 5-tuple of integers (grid-width and a pair of integer points in the plane) to a list of pairs consisting of some information type and a (blue) edge in the WSPD. Let $\mathcal{K} \subset \mathbb{Z} \times \mathbb{Z}^2 \times \mathbb{Z}^2$ denote the set of actual keys for $\mathcal{H}$. Note that the size of set $\mathcal{K}$ is $\mathcal{O}(n)$. Using universal hashing [CW79] this data structure has constant expected access time.

Preprocessing

```
– For every blue edge connecting clusters $(A, B)$ in the split tree do the following
    • $c_A \leftarrow \text{center}(R(A))$, $c_B \leftarrow \text{center}(R(B))$
    • $w \leftarrow |c_A c_B|/s$
    • $\widetilde{w} \leftarrow 2^{\lceil \log w \rceil}$
    • $\widetilde{c_A} \leftarrow \text{snap}(c_A, \widetilde{w})$
    • $\widetilde{c_B} \leftarrow \text{snap}(c_B, \widetilde{w})$
    • Append $((I_{(A,B)}, (A, B)))$ to $\mathcal{H}[\widetilde{w}, \widetilde{c_A}, \widetilde{c_B}]$
```

That is, we determine $\widetilde{w}$, which is the next power of two of the largest possible radius w of the two minimum enclosing disks of $R(A)$ and $R(B)$, respectively (largest possible such that (A, B) can still be a pair of the WSPD). We use $\widetilde{w}$ as width of a grid on which we snap the centers c_A and c_B. The snapped grid positions together with the grid-width are used as keys for the hash table datastructure.

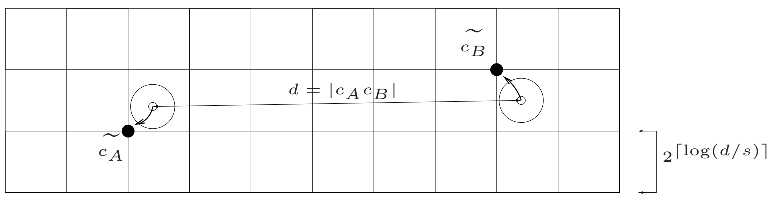

Fig. 6. *Cluster centers c_A and c_B are snapped to closest grid points $\widetilde{c_A}$ and $\widetilde{c_B}$*

Look at Figure 6 for a sketch of the preprocessing routine for one cluster pair (A, B). Clearly this preprocessing step takes linear time in the size of the WSPD. So given a query pair (p, q), how to retrieve the information $I_{(A,B)}$ stored with the unique cluster pair (A, B) with $p \in A$ and $q \in B$?

The idea is that given p and q, the grid-width $\widetilde{w}$ of the grid where c_A and c_B of the cluster pair (A, B) have been snapped to can only be one of $2^{\lceil \log |pq|/s \rceil - 1}$, $2^{\lceil \log |pq|/s \rceil}$, and $2^{\lceil \log |pq|/s \rceil + 1}$. We can afford to try out each of those grid-widths.

```
QUERY($\boldsymbol{p}, \boldsymbol{q}$)
  – $w' \leftarrow |pq|/s$
  – $\widetilde{w_1} \leftarrow 2^{\lceil \log w' \rceil - 1}$
  – $\widetilde{w_2} \leftarrow 2^{\lceil \log w' \rceil}$
  – $\widetilde{w_3} \leftarrow 2^{\lceil \log w' \rceil + 1}$
  – for grid-widths $\widetilde{w_i}$, $i = 1, 2, 3$ and adjacent grid-points $\widetilde{c_p}, \widetilde{c_q}$ of $p$ and $q$ respectively
      • Inspect all items $(I_{(A,B)}, (A, B))$ in $\mathcal{H}[\widetilde{w_i}, \widetilde{c_p}, \widetilde{c_q}]$
          ∗ if $p \in A$ and $q \in B$ return $I_{(A,B)}$
```

In this description we call a grid-point $\widetilde{g}$ *adjacent* to a point p if $|\widetilde{g}p|_x, |\widetilde{g}p|_y < \frac{3}{2}\widetilde{w_i}$, where $|\cdot|_{x/y}$ denotes the horizontal/vertical distance. Clearly there are at most 9 *adjacent* points for any point p in a grid of width $\widetilde{w_i}$ (see Figure 7).

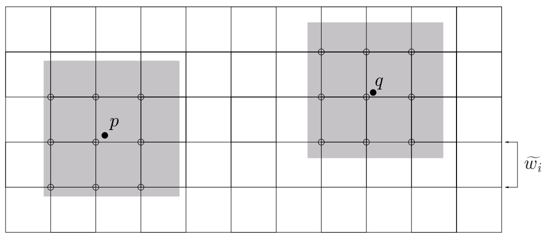

Fig. 7. *Adjacent points of p and q are grid points (hollow circles) inside the rectangles centered in p and q of side-length $3\widetilde{w_i}$.*

In the remainder of this section we will show that this query procedure outputs the correct result (the unique $I_{(A,B)}$ with $p \in A$ and $q \in B$ such that (A, B) is a blue edge in the WSPD) and requires only constant time. We denote by $\widetilde{w} = 2^{\lceil \log |c_A c_B|/s \rceil}$ the grid-width of the grid, to which the cluster centers c_A, c_B of the cluster pair (A, B) we are looking for were snapped.

Lemma 8. *For $s > 4$, we have $\widetilde{w_i} = \widetilde{w}$ for some $i \in \{1, 2, 3\}$.*

Proof. As A and B are well-separated and $p \in A$ and $q \in B$, we know that $|c_A p|, |c_B q| < |c_A c_B|/s$. Therefore $|c_A c_B|(1 - 2/s) < |pq| < |c_A c_B|(1 + 2/s)$ and hence

$$\lceil \log \frac{|c_A c_B|(1 - 2/s)}{s} \rceil < \lceil \log \frac{|pq|}{s} \rceil < \lceil \log \frac{|c_A c_B|(1 + 2/s)}{s} \rceil$$

Using the fact that $s > 4$ we obtain

$$\lceil \log \frac{|c_A c_B|}{s} - 1 \rceil < \lceil \log \frac{|pq|}{s} \rceil < \lceil \log \frac{|c_A c_B|}{s} + 1 \rceil$$

and therefore

$$\lceil \log \frac{|c_A c_B|}{s} \rceil - 1 < \lceil \log \frac{|pq|}{s} \rceil < \lceil \log \frac{|c_A c_B|}{s} \rceil + 1$$

which proves the Lemma. □

This Lemma says that the correct grid-width as determined by c_A and c_B is amongst the grid-widths tried by the query procedure.. Furthermore for any given grid-width and a pair of query points p and q, there are at most $9 \cdot 9 = 81$ pairs of adjacent grid points to inspect. We still need to argue that given the correct grid-width $\widetilde{w}$, the correct pair of grid points $(\widetilde{c_A}, \widetilde{c_B})$ is amongst these ≤ 81 possible pairs of grid points that are inspected.

Lemma 9. *For $\widetilde{w_i} = \widetilde{w}$, $\widetilde{c_A}$ and $\widetilde{c_B}$ are amongst the inspected grid-points.*

Proof. Without loss of generality, it suffices to show that $\widetilde{c_A}$ is adjacent to p. Clearly we have $|c_A p| < \frac{|c_A c_B|}{s} \leq \widetilde{w}$, so in particular $|c_A p|_x, |c_A p|_y < \widetilde{w}$. Furthermore $|c_A \widetilde{c_A}|_x, |c_A \widetilde{c_A}|_y, \leq \widetilde{w}/2$ and hence $|\widetilde{c_A} p|_{x/y} < \frac{3}{2}\widetilde{w}$, i.e. c_A is *adjacent* to p. □

The last thing to show is that only a constant number of cluster pairs (A, B) can be rounded during the preprocessing phase to a specific pair of grid positions $(\widetilde{g_1}, \widetilde{g_2})$ in a grid of grid-width $\widetilde{w}$ and therefore we only have to scan a list of constant size that is associated with $(\widetilde{w}, \widetilde{g_1}, \widetilde{g_2})$ in the hash table data structure. Before we can prove this, we have to cite a following "packing" Lemma from the original work of Callahan and Kosaraju on the WSPD [CK92].

Lemma 10 (CK92). *Let C be a d-dimensional cube and let $S = \{A_1, \ldots, A_l\}$ be a set of nodes in the split tree such that $A_i \cap A_j = \emptyset$ and $l_{\max}(p(A_i)) \geq l(C)/c$ and $R(A_i)$ overlaps C for all i. Then we have $K(c, d) \leq (3c+2)^d$, where $K(c, d)$ denotes the maximum possible value for $|S|$.*

Here $p(A)$ denotes the parent of a node A in the split tree, $l_{\max}(A)$ the longest side of the minimum enclosing rectangle $R(A)$.

Lemma 11. *Consider a WSPD of a point set P with separation constant $s > 4$, grid width $\widetilde{w}$ and a pair of grid points $(\widetilde{g_1}, \widetilde{g_2})$. The number of cluster pairs (A, B) such that c_A and c_B are rounded to $(\widetilde{g_1}, \widetilde{g_2})$ is $\mathcal{O}(1)$.*

Proof. In the following we will establish a lower bound on $r(p(A))$ such that we can apply Lemma 10, as $r(p(A)) \leq l_{\max}(p(A))$. Since $(p(A), B)$ is not a blue edge in the WSPD we have $r(p(A)) > \frac{|c_{p(A)} c_B|}{s}$. Furthermore $|c_{p(A)} c_B| \geq |c_A c_B| - r(p(A))$ and hence using the fact that $\frac{s\widetilde{w}}{2} < |c_A c_B| \leq s\widetilde{w}$ we obtain the following lower bound

$$r(p(A)) > \frac{s\widetilde{w}}{2 + 2s} = \frac{\widetilde{w}}{2 + 2/s}$$

So for $s > 4$ we obtain $l_{\max}(p(A)) > \frac{2}{5}\widetilde{w}$. Before we can apply Lemma 10, observe that if the clusters around $\widetilde{g_1}$ which are paired with some clusters around $\widetilde{g_2}$ are not disjoint (i.e. one is the parent of another), we can just refine those clusters until no cluster is the parent of another, this will only increase the number of links to "the other side". But applying Lemma 10 for the cube centered at $\widetilde{g_1}$ with side-length $\widetilde{w}$ and $c = 5/2$ bounds even this larger number of clusters by $(3 \cdot 5/2 + 2)^2 < 91$. As the same bound holds for the clusters around $\widetilde{g_2}$, the number of cluster pairs that might be assigned to $(\widetilde{g_1}, \widetilde{g_2})$ is less than $91 \cdot 91 < 9000$ and therefore $\mathcal{O}(1)$. □

Putting everything together we get the main theorem of this section:

Theorem 6. *Given a well-separated pair decomposition of a point set P with separation constant $s > 4$, we can construct a data structure in space $\mathcal{O}(n \cdot s^2)$ and construction time $\mathcal{O}(n \cdot s^2)$ such that for any pair of points (p, q) in P we can determine the unique pair of clusters (A, B) that is part the well-separated pair decomposition with $p \in A$, $q \in B$ in constant time.*

Together with the results of the previous section we obtain the following main result of our paper:

Theorem 7. *Given a set of points $P \subset \mathbb{R}^2$, a distance function $\omega : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$ of the form $\omega(p, q) = |pq|^\delta + C_p$, for some constants $\delta \geq 1$, $k \geq 2$ and nonnegative offset cost C_p, we can construct a data structure of size $\mathcal{O}(\frac{1}{\epsilon^2} \cdot n)$ in preprocessing time $\mathcal{O}(\frac{1}{\epsilon^4} \cdot n \log n + \frac{1}{\epsilon^6} \cdot n)$ such that for any query (p, q) from P, a pointer to $(1+\epsilon)$-approximate lightest k-hop path from p to q can be obtained in $\mathcal{O}(1)$ time which does not depend on δ, ϵ, k. Reporting the entire path takes an additional $O(k)$ time.*

We also remark that there are techniques to maintain the well-separated pair decomposition dynamically, and so our whole construction can be made dynamic as well (see [CK95]).

4 Computing the Shortest Paths without Hop Restrictions

In this section we consider the case of computing the shortest path without any restrictions in the number of hops used for radio networks when $\delta = 2$. Since all edge weights are nonnegative, we can use the dynamic programming algorithm from Theorem 1, which guarantees a running time of $\mathcal{O}(n^2 \log n)$. This is even worse than applying Dijkstra's algorithm to the complete graph. Therefore, we use a different shortest path algorithm. It is similar to Dijkstra's algorithm in that it settles nodes in order of the distance from the source. But edges are never explicitly relaxed. Instead a powerful geometric data structure replaces both edge relaxation and the priority queue of Dijkstra's algorithm.

Theorem 8. *In a complete geometric graph defined by n points in the plane with edge weights $C_p + |pq|^2$, the single source shortest path problem can be solved in $\mathcal{O}(n^{1+\epsilon})$ time and space, for any constant $\epsilon > 0$.*

Proof. Let $\mu(v)$ denote the length of a shortest path from source s to $v \in V$. During the execution of the algorithm each node is in one of two possible states, settled (red) and unsettled (blue). Initially all nodes except the source node are unsettled. Nodes become settled in order of increasing $\mu()$ values. At the time a node becomes settled, its $\mu()$ value is known. Let R and B denote the set of settled (red) and unsettled (blue) nodes respectively. In each iteration, we determine a blue node with smallest distance to the source:

$$v = \arg\min_{b \in B} \min_{r \in R} \{\mu(r) + |rb|^2 + C_r\} \tag{4}$$

As in Section 2.1, we use an embedding of the points into $\mathbb{R}^3$ which allows us to find the node v that minimizes the above expression by solving a closest bichromatic point problem. We define two functions $f_0, f_\mu : P \longrightarrow \mathbb{R}^3$ by

$$f_\mu(r) = (x_r, y_r, \sqrt{\mu(r) + C_r}) \text{ and } f_0(b) = (x_b, y_b, 0)$$

For any red-blue pair $(r, b) \in R \times B$, we have $|f_\mu(r) f_0(b)|^2 = |rb|^2 + \mu(r) + C_r$ according to Pythagoras. Hence, the closest bichromatic pair with the sets $\{f_\mu(r) \mid r \in R\}$ and $\{f_0(b) \mid b \in B\}$ gives a solution to Equation 4. By slightly abusing notation, the algorithm can be stated as follows.

```
$B \leftarrow V \backslash \{s\}$, $R \leftarrow \{s\}$, $d[s] \leftarrow 0$;
while ($B \neq \emptyset$)
  Find closest bichromatic pair $(r, b)$ in $\{f_d(r) \mid r \in R\}$ and $\{f_0(b) \mid b \in B\}$.
  $R \leftarrow R \cup \{b\}$, $B \leftarrow B \backslash \{b\}$, $d[b] \leftarrow |f_d(r) f_0(b)|^2$;
```

The dynamic bichromatic closest points problems we have to solve are of special type: all blue points are lying in the xy-plane and the distance function is simply the Euclidean distance. This enables us to speed up the computation compared to the general 3D dynamic closest pair problem.

A general technique for solving the dynamic bichromatic closest pair problem is given by Eppstein [Epp95], who reduces the problem to the *dynamic nearest neighbors problem* for dynamically changing sets of sites R and B. For blue sites B the queries are red points ($q \in R$) and vice versa. Since in our application the blue set is essentially a 2D point set, we distinguish two types of nearest neighbor queries:

1. Sites S are restricted to the xy-plane, query points unrestricted.
2. Queries q are restricted to the xy-plane, sites are unrestricted.

We show that these special instances of the 3D problem can be regarded as 2D nearest neighbor problems. First consider finding the nearest blue point for a given red query point $q = (q_x, q_y, q_z)$. Since all blue points lay in the xy plane we can simply query with point $q' = (q_x, q_y, 0)$. It is easy to verify that the nearest neighbor for q' is also the nearest neighbor for q.

The second type of nearest neighbor queries was already considered in Section 2.1 for a static set of sites, where we used power diagrams. Now we have to deal with a dynamic set of points. We will exploit the well-known relation between power diagrams and the upper envelope of 3D hyperplanes. In particular, finding the nearest site is equivalent to finding the first hyperplane that is intersected by a query ray (see [AM92]). Agarwal and Matoušek describe algorithms for dynamic ray shooting problems and state the following lemma.

Lemma 12 ([AM92]). *Given a convex polytope in $\mathbb{R}^3$ described as the intersection of n half-spaces, one can process it in time $\mathcal{O}(n^{1+\epsilon})$ into a data structure of size $\mathcal{O}(n^{1+\epsilon})$, so that the first point of the polytope boundary hit by a query ray can be determined in $\mathcal{O}(\log^5 n)$ time. The data structure can be maintained dynamically in amortized time $\mathcal{O}(n^\epsilon)$ per insert/delete operation.*

Combined with the result of Eppstein ([Epp95], Theorem 1) for maintaining the bichromatic closest pair, this proves Theorem 8. □

Note that in [CE01], the lower envelope of a set of bivariate functions is used to solve the nearest neighbor problem. In contrast, we considered a more restrictive cost function and maintain a set of linear objects, namely planes, which leads to a better running time.

Acknowledgments

We would like to thank Edgar Ramos for contributing valuable geometric know-how and Sören Laue for helpful conversations related to proof of Lemma 5. Many thanks to Alexandra Zhilyakova for creating the Figure 1.

References

[Aur87] F. Aurenhammer. Power Diagrams: Properties, Algorithms and Applications. *SIAM J. Comput., 16(1):78-96, 1987*

[AM92] P. D. Agarwal and J. Matousek. Ray shooting and parametric search. In N. Alon, editor, *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 517–526, Victoria, B.C., Canada, May 1992. ACM Press.

[AM00] S. Arya and D. M. Mount:*Approximate range searching*, Computational Geometry: Theory and Applications, (17), 135-152, 2000

[AM98] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, A. Wu: *An optimal algorithm for approximate nearest neighbor searching*, Journal of the ACM, 45(6):891-923, 1998

[BSS02] R. Beier, P. Sanders, N. Sivadasan. Energy Optimal Routing in Radio Networks Using Geometric Data Structures. Proc. of the 29th Int. Coll. on Automata, Languages, and Programming, 2002.

[Bel57] R. Bellman. Dynamic Programming. Princeton Univ. Press, Princeton, N.J., 1957.

[BKOS] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications.* Springer-Verlag, Berlin Heidelberg, 2., rev. ed. edition, 2000.

[CK92] P.B. Callahan, S.R. Kosaraju. A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields. Proc. 24th Ann. ACM Symp. on the Theory of Computation, 1992

[CK95] P.B. Callahan, S.R. Kosaraju: *Algorithms for Dynamic Closest Pair and n-Body Potential Fields*, Proc. 6th Ann. ACM-SIAM Symp. on Discrete Algorithm, 1995

[CW79] J.L. Carter and M.N. Wegman. Universal Classes of Hash Functions. *Journal of Computer and System Sciences, 18(2):143–154, 1979*

[CE01] T. Chan and A. Efrat. Fly cheaply: On the minimum fuel consumption problem. *Journal of Algorithms*, 41(2):330–337, November 2001.

[Dij59] E. W. Dijkstra. A note on two problems in conexion with graphs. In Numerische Matematik, 1:269-271, 1959.

[EH98] A. Efrat, S. Har-Peled. Fly Cheaply: On the Minimum Fuel-Consumption Problemma. Proc. 14th ACM Symp. on Computational Geometry 1998.

[Epp95] D. Eppstein. Dynamic euclidean minimum spanning trees and extrema of binary functions. *Disc. Comp. Geom.*, 13:111–122, 1995.

[For56] L. R. Ford. Network Flow Theory. Report P-923, The Rand Corporation, Santa Monica, Cal, 1956.

[FMS03] S. Funke, D. Matijevic, P. Sanders. Approximating Energy Efficient Paths in Multi-Hop Networks. Proc. of 11th European Symposium on Algorithms (ESA), number 2832 in LNCS. Springer, 2003.

[GIV01] A. Goel, P. Indyk, K. Varadarajan. Reductions Among High Dimensional Proximity Problems. Proc. of 10th Symposium on Discrete Algorithms (SODA), 769–778, 2001.

[MN90] K. Mehlhorn, S. Näher. Dynamic Fractional Cascading. Algorithmica (5), 1990, 215–241

[NOZ01] K. Nakano, S. Olariu, and A. Y. Zomaya. Energy-efficient permutation routing in radio networks. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):544–557, 2001.

[Pat00] D. Patel. Energy in ad-hoc networking for the picoradio. Master's thesis, UC Berkeley, 2000.

[Rab00] J. M. Rabaey et al. Picoradio supports ad hoc ultra-low power wireless networking. *IEEE Computer Magazine*, pages 42–48, July 2000.

[Rap96] T. S. Rappaport. *Wireless Communication.* Prentice Hall, 1996.

[TZ01] M.Thorup and U.Zwick. Approximate Distance Oracles *Proc. of 33rd Symposium on the Theory of Computation 2001.*