Medium-Space Algorithms for Inverse BWT[⋆]

Juha Kärkkäinen[1] and Simon J. Puglisi[2]

[1] Department of Computer Science, University of Helsinki, Finland
juha.karkkainen@cs.helsinki.fi
[2] School of Computer Science and Information Technology,
Royal Melbourne Institute of Technology, Australia
simon.puglisi@rmit.edu.au

Abstract. The Burrows–Wheeler transform is a powerful tool for data compression and has been the focus of intense research in the last decade. Little attention, however, has been paid to the inverse transform, even though it is a bottleneck in decompression. We introduce three new inversion algorithms with improved performance in a wide range of the space-time spectrum, as confirmed by both theoretical analysis and experimental comparison.

1 Introduction

The Burrows–Wheeler transform (BWT) [2,1] is an invertible transformation of a text that has a central role in some of the best data compression methods. The transform itself performs no compression — the result is just a permutation of the text — but the transformed text is easier to compress using simple and fast methods [15]. Much effort has gone into developing efficient algorithms for the forward transform, largely owing to its close relation to constructing the suffix array [17] and compressed text indexes [16]. The less studied problem of inverting the transform is the subject of this paper.

The inverse transform is a bottleneck in decompression and thus needs to be fast, particularly in applications requiring frequent decompression such as on-the-fly disk compression. The space requirement is also an issue: a typical, fast implementation requires 5 times the space of the text. As already proposed in the original paper [2], the text can be broken into smaller blocks, each of which is compressed separately. However, a large block size is preferable because it allows better compression (see e.g. [4]). Furthermore, the block size is determined during the forward transform, possibly on a machine with more memory or using a space efficient algorithm (e.g. [11]), and the inverse transform is impossible unless a sufficiently space-efficient algorithm is available.

The key operation in the inversion is a rank query:

$$\mathrm{rank}(j) \equiv \left|\{i \mid i < j \text{ and } L[i] = L[j]\}\right| ,$$

[⋆] This work is supported by the Academy of Finland grant 118653 (ALGODAN) and by the Australian Research Council.

where L is the transformed text. A single scan of L is sufficient to answer all distinct rank queries in the sequential order, but during the inversion they are needed in a different order. By tabulating the answers, we obtain a simple, linear time inversion algorithm, already described in [2]. There are a few variations but all of them need at least $n \log n$ bits of space for the tabulated answers or something equivalent. We call these *large-space algorithms*.

Space-efficient data structures for rank queries are widely used with compressed text indexes [16], but the query needed there is of a more general form:

$$\text{rank}(c, j) \equiv \left|\{i \mid i < j \text{ and } L[i] = c\}\right| .$$

We call this the *general rank query* as opposed to the *special rank query* needed in inverse BWT algorithms. Obviously, any data structure for general rank queries can be used for special rank queries, so using the techniques from compressed text indexes, we obtain space-efficient algorithms for inverse BWT. Many of these algorithms need at most $n \log \sigma + o(n \log \sigma)$ bits of space, where σ is the size of the alphabet. This is only slightly more than needed for the text itself — and sometimes even less by way of compression — but this comes at a significant cost in query time, especially in practice. We call these *small-space algorithms*.

The focus of this paper is on *medium-space algorithms* that are between large-space and small-space algorithms with respect to both time and space complexity. The key characteristic of medium-space algorithms is the tabulation of *partial* answers to all special rank queries.

Related work. Seward [19] describes several large-space BWT inversion algorithms, among them the original algorithm from [2], and compares them experimentally. One of the algorithms, `mergedTL`, is the fastest known algorithm in practice. Seward also has two algorithms in the small-space category, but they are not competitive either in theory or in practice.

The only previous medium-space algorithm we are aware of is by Lauther and Lukovszki [13]. They also propose two small-space algorithms and provide experimental results for two of their algorithms. They identify the central role of the special rank query but not its relation to the general rank query.

Ferragina, Gagie and Manzini [3] have recently described an external memory algorithm for the inversion. It is, however, rather complicated and unlikely to be competitive except when external memory algorithms are the only option.

There is a large body of research on space-efficient data structures for (general) rank queries in the area of compressed text indexes and succinct data structures. The theoretically best results on BWT inversion achievable using those techniques are reported at the bottom of Table 1.

Our contribution. We introduce three new medium-space algorithms for BWT inversion, offering improved space–time tradeoffs, including the most space-efficient linear-time algorithm for large alphabets. The time and space complexities are shown in Table 1. The table also shows our improved analysis of the only previous medium-space algorithm in [13]. Experimental results show that the favorable tradeoff properties extend to practice too.

Table 1. Time and space complexities of BWT inversion algorithms. The sections correspond to large-, medium- and small-space algorithms. The space complexities exclude the space for input, output and $\mathcal{O}(\sigma \log n)$-bit data structures.

space (bits/symbol)	time per symbol	comment
$\lceil \log n \rceil + \lceil \log \sigma \rceil$	$\mathcal{O}(1)$	`mergeTL` in [19]
$\lceil \log n \rceil$	$\mathcal{O}(\log \sigma)$	`indexF` in [19]
$\log \lceil \log n \rceil + \log \sigma + \lceil \log \sigma \rceil$	$\mathcal{O}(1)$	[13]
$1 + \log b + \lceil \log \sigma \rceil$	$\mathcal{O}(\log(n/b)$ $-H_0 + b\sigma/n)$	this paper $\lceil \log n \rceil \leq b \leq n/\sigma$
$2 + \log\left(\lceil \log n \rceil + 3\lceil \log \sigma \rceil\right)$ $+ \log \sigma + \frac{2 \log \log n}{\log n}$	$\mathcal{O}(1)$	this paper
$1 + \log b + \log \sigma$	$\mathcal{O}(\log(n/b) - \log \sigma)$	this paper $2(1 + \lceil \log n \rceil) \leq b \leq n/\sigma$
$\lceil \log \sigma \rceil + (\sigma \log n)/b$	$\mathcal{O}(b)$	[13]
$\lceil \log \sigma \rceil + \frac{\sigma}{b_1}\left(\log b_1 + \frac{\log n}{b_2}\right)$	$\mathcal{O}(b_1 + b_2)$	[13]
$H_k + \mathcal{O}\left(\frac{\log \sigma \log \log n}{\log n} + \frac{\sigma^{k+1} \log n}{n}\right)$	$\mathcal{O}(1 + \frac{\log \sigma}{\log \log n})$	[5,14]
$\log \sigma + \mathcal{O}(\frac{\log \sigma}{\log \log \sigma})$	$\mathcal{O}(\log \log \sigma)$	[6]

Perhaps of independent interest is the identification of the special rank query as an operation of interest, separate from the general rank query. The separate nature is illustrated by the fact that extending the techniques used by the large- and medium-space algorithms to general rank queries would blow up the space by factor σ, which is usually too much. We note that, besides inverse BWT, the locate and display procedures over BWT-based compressed indexes (see [16]) perform repeated special rank queries.

2 Preliminaries

Let $S = S[0..n] = S[0]S[1]\dots S[n]$ be a string of $n+1$ symbols or characters. The first n characters of S are drawn from an ordered alphabet Σ, and the final character $S[n]$ is a special "end of string" symbol, \$, distinct from and lexicographically smaller than all the other symbols. We assume that the symbols in Σ are encoded with the integers $\{0, 1, .., \sigma - 1\}$ in an order preserving way.

For any $i \in 0..n$, the string $S[i..n]S[0..i-1]$ is a *rotation* of S. Let $\mathcal{M}$ be the $(n+1) \times (n+1)$ matrix whose rows are all the rotations of S in lexicographic order. Let F be the first and L the last column of $\mathcal{M}$, both taken to be strings of length $n+1$. The string L is the ***Burrows–Wheeler transform*** of S. An example is given in Fig. 1. Note that F and L are permutations of S.

For a string X, integers $j, r \in \{0, \dots, |X| - 1\}$ and a symbol c, define the following functions:

$$\begin{aligned} \text{access}_X(j) &\equiv X[j] \\ \text{rank}_X(j) &\equiv \left|\{i \mid i < j \text{ and } X[i] = X[j]\}\right| \end{aligned}$$

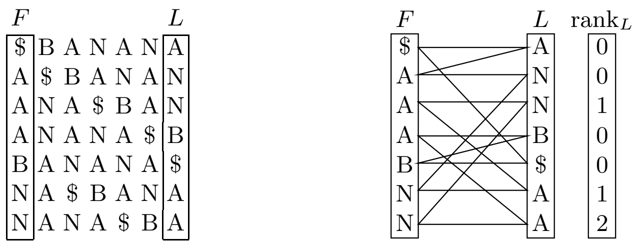


Fig. 1. BWT matrix $\mathcal{M}$ and inverse BWT permutation for text $S = \text{BANANA\$}$

Inverse BWT in forward order

```
1: construct $F$ by sorting $L$
2: $j \leftarrow \text{select}_L(\$, 0)$
3: for $i \leftarrow 0$ to $n$ do
4:     $S[i] \leftarrow c \leftarrow \text{access}_F(j)$
5:     $r \leftarrow \text{rank}_F(j)$
6:     $j \leftarrow \text{select}_L(c, r)$
```

Inverse BWT in reverse order

```
1: construct $F$ by sorting $L$
2: $j \leftarrow \text{select}_L(\$, 0)$
3: for $i \leftarrow n$ downto $0$ do
4:     $S[i] \leftarrow c \leftarrow \text{access}_L(j)$
5:     $r \leftarrow \text{rank}_L(j)$
6:     $j \leftarrow \text{select}_F(c, r)$
```

Fig. 2. Two abstract algorithms for the inverse Burrows–Wheeler transform

$$\text{select}_X(c, r) \equiv \begin{cases} j & \text{if } X[j] = c \text{ and } \text{rank}_X(j) = r \\ \text{undefined} & \text{if there is no such } j \end{cases}$$

The notation $\text{access}_X(j)$ is used instead of $X[j]$ when X might be stored in a form that does not support trivial character access.

Two abstract inversion algorithms are given in Fig. 2. The first (left-hand side) algorithm constructs S from the beginning to the end and the second in the reverse order. Both algorithms follow the same unicyclic permutation but in different directions. An example of the permutation is shown in Fig. 1. To obtain concrete algorithms, we need to define the implementation of the operations access, rank, and select.

3 Basic Large-Space Algorithms

Of the two abstract algorithms in Fig. 2, we will focus on the reverse order algorithm, as its operations are easier to implement and faster in practice. All the algorithms mentioned in this paper are based on it.

Another feature shared by all the algorithms is the implementation of select_F based on the special nature of F. The string F contains the characters of S in sorted order and all copies of the same symbol are grouped together. For any symbol c, let $C[c]$ be the position of the first occurrence of c in F. We can implement select_F as

$$\text{select}_F(c, r) = C[c] + r \ .$$

The array C can be easily computed by scanning L.

The difference between various algorithms is the implementation of access_L and rank_L. We will describe next the algorithm from the seminal paper by Burrows and Wheeler [2]. They store L explicitly, making access_L trivial. The values $\text{rank}_L(j)$ are stored in a table $R[0..n]$, which can be computed by scanning L while keeping account of the number of occurrences of each symbol.

The algorithm runs in linear time and needs $n(\lceil\log n\rceil + \lceil\log\sigma\rceil) + (\sigma + \mathcal{O}(1))\lceil\log n\rceil$ bits of space. It would be very fast in practice, but for cache misses. In the main loop, the sequence of accesses to L and R is essentially random with a high likelihood of a cache miss for each access. Seward [19] describes an optimized version that replaces the arrays L and R with a single array LR that stores both values. This can reduce the number of cache misses to a half, leading to a significant improvement in speed. This algorithm, which we call Algorithm `LR` (`mergeTL` in [19]), is the fastest known algorithm for BWT inversion. It is also the starting point for our medium-space algorithms.

4 Basic Medium-Space Algorithms

In this section we describe two simple medium-space algorithms. One of them is by Lauther and Lukovszki [13] and one is new.

Both algorithms modify Algorithm `LR` by storing only partial information about ranks in R (i.e., in the R-fields of the array LR). Every position $j \in \{0,\ldots,n\}$ is associated with a nearby *reference point* $\text{ref}(j) \in \{0,\ldots,n\}$, and

$$R[j] = \text{rank}_L(j) - \text{rank}_L(L[j], \text{ref}(j)) \ .$$

Now we can compute a rank query as $\text{rank}_L(j) = \text{rank}_L(L[j], \text{ref}(j)) + R[j]$. The difference between the two algorithms is the choice of reference points and the computation of $\text{rank}_L(L[j], \text{ref}(j))$.

4.1 Algorithm `LR-B`

The first algorithm is by Lauther and Lukovszki [13]. We provide an improved analysis.

Divide R into $\lceil(n+1)/b\rceil$ blocks of size b. Every position in a block is associated with the same reference point, which is the center of the block. In other words, the reference points are the positions $b/2, b+b/2, 2b+b/2, \ldots$. As a small twist to the basic scheme, if j is in the first half of a block, i.e., if $j < \text{ref}(j)$, we set

$$R[j] = \text{rank}_L(L[j], \text{ref}(j)) - \text{rank}_L(j) - 1 \ .$$

Otherwise, i.e., if $j \geq \text{ref}(j)$, we use the basic scheme and set

$$R[j] = \text{rank}_L(j) - \text{rank}_L(L[j], \text{ref}(j)) \ .$$

Now all the values in R are in the range $[0, b/2-1]$ and can be stored using $\lceil\log b\rceil - 1$ bits. The ranks at the reference points are stored in a two-dimensional array R_{ref}, i.e., for all $c \in \Sigma$ and $j \in \{0,\ldots,\lceil(n+1)/b\rceil - 1\}$,

$$R_{\text{ref}}[c, j] = \text{rank}_L(c, b/2 + jb) \ .$$

We need at most $\sigma(n/b+1)\lceil \log n \rceil$ bits for the array R_{ref}.

The following theorem summarizes the properties of the algorithm. All proofs are omitted here due to lack of space and are provided in the full paper.

Theorem 1. *Setting $b = 2^k$ for $k = \lfloor \log(\sigma \lceil \log n \rceil) \rfloor$, Algorithm `LR-B` computes the inverse Burrows-Wheeler transform in $\mathcal{O}(n)$ time using at most*

$$n(\log \lceil \log n \rceil + \log \sigma + \lceil \log \sigma \rceil) + \mathcal{O}(\sigma \lceil \log n \rceil)$$

bits of space.

4.2 Algorithm `LR-I`

In our new algorithm, reference points are separate for each symbol of the alphabet. For a symbol c, the reference points are at every bth occurrence of c, i.e., at positions $\text{select}_L(c,0), \text{select}_L(c,b), \text{select}_L(c,2b), \ldots$. A position j is assigned to the closest preceding reference point for the symbol $L[j]$, i.e.,

$$\text{ref}(j) = \text{select}_L(L[j], b\lfloor \text{rank}_L(j)/b \rfloor) \ .$$

The array R is as in the basic scheme, i.e., $R[j] = \text{rank}_L(j) - \text{rank}_L(L[j], \text{ref}(j))$, and we need $\lceil \log b \rceil$ bits for each entry. The reference points for a symbol c are stored in an array I_c, i.e., $I_c[i] = \text{select}_L(c, ib)$. The arrays I_c, $c \in \Sigma$, can be seen as sparse inverted lists for the symbols. The total space for them is $n\lceil \log n \rceil / b + \mathcal{O}(\sigma \log n)$ bits. To compute $\text{rank}_L(j)$, we binary search $I_{L[j]}$ to find i such that $I_{L[j]}[i] \le j < I_{L[j]}[i+1]$, and then $\text{rank}_L(j) = ib + R[j]$.

Unlike `LR-B`, Algorithm `LR-I` offers a space-time tradeoff as shown by the following result.

Theorem 2. *Let $b = 2^k$ for an integral k. If $k = \lfloor \log \lceil \log n \rceil \rfloor$, the space requirement of Algorithm `LR-I` is at most*

$$n(1 + \log \lceil \log n \rceil + \lceil \log \sigma \rceil) + \mathcal{O}(\sigma \log n) \ .$$

For $\lfloor \log \lceil \log n \rceil \rfloor < k \le \log(n/\sigma)$, the space requirement is at most

$$n(1 + k + \lceil \log \sigma \rceil) + \mathcal{O}(\sigma \log n) \ .$$

The time complexity is $\mathcal{O}(n(\log(n/b) - H_0) + b\sigma)$, where $H_0 \le \log \sigma$ is the zeroth order empirical entropy of S (see Section 5).

5 Variable-Length Encoding

In this section, we show how to improve the algorithms of the previous section using variable-length encoding.

For a string X, let Σ_X be the set of symbols occurring in X, let $|X_c|$ be the number of occurrences of a symbol c in X, and let $f_X(c) = |X_c|/|X|$ be the frequency of c. The *zeroth order empirical entropy* of X is

$$H_0(X) = \sum_{c \in \Sigma_X} f_X(c) \log(1/f_X(c)) \ .$$

A *canonical prefix code* [18] for X is characterized by a non-decreasing sequence $\ell = (\ell_1, \ldots, \ell_{|\Sigma_X|})$ of positive, integral code lengths satisfying Kraft's inequality: $\sum_{i=1}^{|\Sigma_X|} 2^{-\ell_i} \leq 1$. The code lengths are assigned to symbols in decreasing order of symbol frequency; let $\ell(c)$ denote the code length of a symbol c. There exists an assignment of binary code words code(c) of $\ell(c)$ bits to each symbol c so that, for every $c, c' \in \Sigma_X$ with $f_X(c) < f_X(c')$,

– code(c) is not a prefix of code(c') (the code is *prefix-free*), and
– code(c) is lexicographically smaller than code(c') (the code is *canonical*).

Let $\ell(X)$ be the encoded length of X for a code ℓ:

$$\ell(X) = \sum_{i=0}^{m-1} \ell(X[i]) = m \sum_{c \in \Sigma_X} f_X(c)\ell(c) \ .$$

For any prefix code, $\ell(X) \geq mH_0(X)$. The equality is achieved with the *fractional* lengths $\ell(c) = \log(1/f_X(c))$. The Huffman code [9] is known to be the optimal code with integral lengths. However, for our purposes, we need a code where the code length of every symbol is close to the fractional optimum, which the Huffman code does not guarantee [12]. Furthermore, with one of our algorithms (`VRL-I`, Section 5.2), we have a strict upper limit h on the code lengths. We will be using the *length-limited rounded code* $\hat{\ell}_X^h$ with

$$\hat{\ell}_X^h(c) = \lceil h - \log(f_X(c)(2^h - \sigma_X) + 1) \rceil \ ,$$

for any integer $h \geq \lceil \log \sigma_X \rceil$. When there is no upper limit, the code is $\hat{\ell}_X = \hat{\ell}_X^\infty$ with $\hat{\ell}_X(c) = \lceil \log(1/f_X(c)) \rceil$. The properties of the code are established in the following lemma.

Lemma 1. *The code lengths $\hat{\ell}_X^h$ define a valid prefix code for X with $\hat{\ell}(c) \leq h$ for all $c \in \Sigma$. Furthermore, for all $c \in \Sigma$,*

$$\hat{\ell}_X^h(c) < \log(1/f_X(c)) + \log(2^h/(2^h - \sigma_X)) + 1 \ ,$$

and if $\log(1/f_X(c)) \geq h$*, then* $\hat{\ell}_X^h(c) = h$.

5.1 Algorithm `VLR-B`

As with `LR-B`, we divide the array LR into blocks of size b. The reference point for all positions in a block is now the beginning of the block (instead of the center).

Let B be a block. We encode the L-fields in B with the unlimited rounded code $\hat{\ell}_B$, and the R-fields using $\lceil \log |B_c| \rceil$ bits for a symbol c. The combined length of the two fields for a symbol c is

$$\lceil \log(1/f_B(c)) \rceil + \lceil \log |B_c| \rceil = \lceil \log(b/|B_c|) \rceil + \lceil \log |B_c| \rceil \leq \lceil \log b \rceil + 1 \ .$$

Thus we need $n(\lceil \log b \rceil + 1)$ bits for the whole LR array.

For each block B, we have a table V with an entry for each symbol c in Σ_B containing three fields

- The e-field has $\lceil \log b \rceil + 1$ bits with code(c) in the beginning and the rest of the field filled with zeros.
- The s-field contains the original code for c using $\lceil \log \sigma \rceil$ bits.
- The r-field is the rank of the symbol c at the reference point, i.e., at the beginning of the block in $\lceil \log n \rceil$ bits.

The table V is ordered by the e-field. Given $LR[j]$, we find the entry in $V[i]$ such that $V[i].e \leq LR[j] < V[i+1].e$. We obtain the symbol $L[j]$ from $V[i].s$ and its rank at the reference point from $V[i].r$. The rank relative to the reference point is $LR[j] - V[i].e$. Thus $\mathrm{rank}_L(j) = V[i].r + (LR[j] - V[i].e)$.

To speed up the search in V, there is another table $U[0..2^q - 1]$, $0 \leq q \leq \lceil \log b \rceil + 1$. The entries of U represent the bitstrings of length q. The entry for a bitstring Q contains a pointer to the first position in V with a code beginning with Q. If a code is shorter than q, say $\hat{\ell}_B(c) < q$, all bitstrings beginning with code(c) point to $V[c]$. We need $\lceil \log \sigma \rceil$ bits for each pointer.

Using U we can short-cut to a good starting point for the search in V. The search itself can be done linearly, and we still obtain a linear-time algorithm as shown by the following theorem.

Theorem 3. *Setting $q = \lfloor \log \sigma \rfloor$ and $b = 2^k$ for $k = \lfloor \log(\sigma(\lceil \log n \rceil + 3\lceil \log \sigma \rceil)) \rfloor$, Algorithm `VLR-B` computes the inverse Burrows-Wheeler transform in $\mathcal{O}(n)$ time using at most*

$$n\left(2 + \log(\sigma) + \log\left(\lceil \log n \rceil + 3\lceil \log \sigma \rceil\right) + \frac{2 \log \log n}{\log n}\right) + \mathcal{O}(\sigma \log n)$$

bits of space.

5.2 Algorithm `VLR-I`

Our final algorithm is a modification of Algorithm `LR-I` to use variable-length fields in the LR array. Each entry in the LR array is $h > \log \sigma$ bits. The L-fields use the length-limited rounded code $\hat{\ell}_L^h$, leaving $h - \hat{\ell}_L^h(c)$ bits for the R field. Thus the reference points are placed at every $b(c)$th occurrence for $b(c) = 2^{h - \hat{\ell}_L^h(c)}$. The decoding of the LR entries is done as in Algorithm `VLR-B`. Otherwise the algorithm works as `LR-I`. The properties are summarized in the following theorem.

Theorem 4. *Let* $h_{\min} = \lfloor 1 + \log \lceil 1 + \log n \rceil + \log \sigma \rfloor$. *When* $h = h_{\min}$, *the space requirement of Algorithm* `VLR-I` *is at most*

$$n(2 + \log \lceil 1 + \log n \rceil + \log \sigma) + \mathcal{O}(\sigma \log n)$$

bits. For $h > h_{\min}$, *the space requirement is at most*

$$n(h + 1) + \mathcal{O}(\sigma \log n)$$

bits. The time complexity is $\mathcal{O}(n(\max(1, \log n - h))$.

6 Experimental Results

For testing we used the files listed in Table 2[1]. All tests were conducted on a 3.0 GHz Intel Xeon CPU with 4Gb main memory and 1024K L2 Cache. The machine had no other significant CPU tasks running. The operating system was Fedora Linux running kernel 2.6.9. The compiler was g++ (gcc version 4.1.1) executed with the -O3 option. The times given are the minima of three runs and were recorded with the standard C `getrusage` function. The memory requirements are sums of the sizes of all data structures as reported by the `sizeof` function.

Table 2. Data sets used for empirical tests. For each type of data (DNA, XML, ENGLISH, PROTEIN) a 100Mb file was used.

Data set name	σ	H_0	mean LCP
XML	97	5.23	44
DNA	16	1.98	31
ENGLISH	239	4.53	2,221
PROTEIN	27	4.20	166

Table 3. Algorithms and their parameter settings. Underlined parameter values indicate that the implementation is optimized for the byte or word alignment provided by those parameters values.

Alg.	Description
`LR`	Algorithm `LR` (Sect. 3) = `mergedTL` in [19] with 32-bit integers
`IF`	`indexF` in [19] with 32-bit integers
`LR-B`	$k \in 5 + \lfloor \log \sigma \rfloor, \ldots, \underline{17}, \underline{25}$
`VLR-B`	$k \in 10, \ldots, 24$
`LR-I`	$k + \lceil \log \sigma \rceil \in 14, \underline{16}, \underline{24}, \underline{32}$
`VLR-I`	$h \in 12, 14, \underline{16}, \underline{24}, \underline{32}$
`LL`	The simple small-space algorithm in [13]
	Blocksizes are powers of two in $[\max(32, \sigma) \ldots \min(2048, 40\sigma)]$
`WT`	A simple algorithm using wavelet tree for rank queries (see text)

[1] Available from `http://pizzachili.dcc.uchile.cl/`

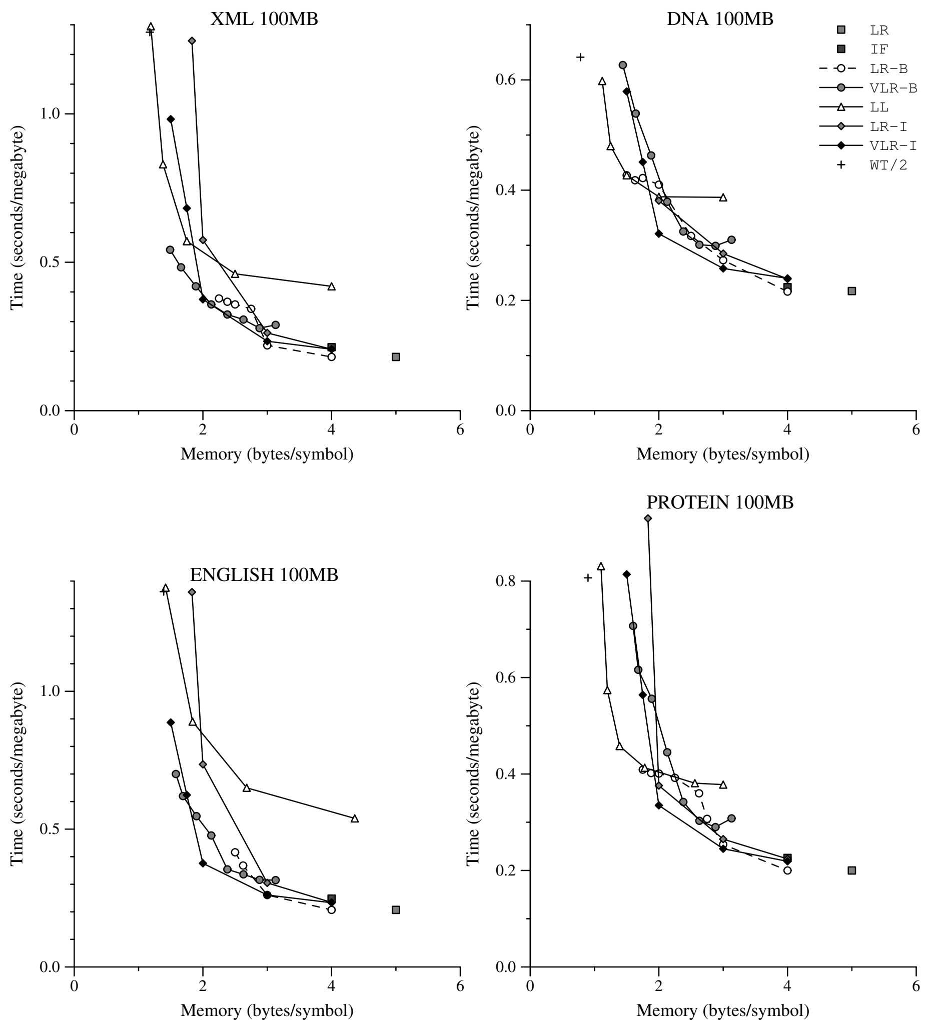


Fig. 3. Time-memory tradeoff for various inversion algorithms. For clarity, *the time shown for* `WT` *is half of the actual time*, which would be far outside the graph.

The focus of the experiments is on the four algorithms described in Sections 4 and 5, but for comparison we also implemented two large-space algorithms by Seward [19] and two simple small-space algorithms, one by Lauther and Lukovszki [13] and one based on the wavelet tree [7], which is a commonly used rank data structure with compressed text indexes [16]. We optimized the wavelet tree implementation for special rank queries and used the method of Vigna [21] (the fastest we know) for bitvector rank queries. For canonical prefix coding, we use the techniques of Turpin and Moffat [20]^2 instead of the technique

^{2} Originally downloaded from `http://ww2.cs.mu.oz.au/~alistair/mr_coder/`

of Sect. 5.1. In all medium- and small-space algorithms, we use $\sigma = |\Sigma_S|$ (see Table 2), which affects arrays of size σ, the height $\lceil \log \sigma \rceil$ of the wavelet tree, and the size $\lceil \log \sigma \rceil$ of the L-field in the LR array for Algorithms `LR-B` and `LR-I`. The algorithms and their parameter settings are summarized in Table 3.

The time and space requirements during BWT inversion are shown in Fig. 3. The times do not include reading the input or writing the output. The input and output are held in memory during the computation but are excluded from space requirements when the algorithm accesses them only sequentially.

All the medium-space algorithms display a fairly smooth space-time tradeoff curve, even the constant-time algorithms with no theoretical tradeoff. This is explained by cache effects. As the LR array (which always dominates the space) gets bigger, the other data structures get smaller and start to fit in the cache.

At the fast end of the space-time tradeoff, `LR-B` matches the speed of the fastest known algorithm, `LR`, in less memory. Note that this parameter setting ($k = 25$) was not implemented or even suggested by Lauther and Lukovszki [13]. The middle area is dominated by the algorithms `VLR-B` and `VLR-I` using variable-length encoding. They reduce the space by a factor of 2–3 compared with `LR` without slowing down by more than a factor of two. The results for the small end are mixed, and anyway should be considered incomplete, since there are many possibilities for improving the small-space algorithms.

7 Concluding Remarks

We have introduced three new algorithms for the BWT inversion and demonstrated, theoretically and experimentally, that they improve the state of the art, particularly in the middle area of the space-time tradeoff spectrum. We are continuing our research by focusing on the extremes of the spectrum.

At the small end, the two-level version of the small-space algorithm by Lauther and Lukovszki [13], and advanced techniques from compressed text indexes such as Huffman-shaped wavelet trees [8] and implicit compression boosting [14] appear promising approaches.

At the large end, we are experimenting with an algorithm that reduces cache misses by taking advantage of repetitions in the text. Another interesting avenue for future work is exploiting properties of modern processors such as parallelism and out-of-order execution [10].

References

1. Adjeroh, D., Bell, T., Mukherjee, A.: The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching. Springer, Heidelberg (2008)
2. Burrows, M., Wheeler, D.J.: A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California (1994)
3. Ferragina, P., Gagie, T., Manzini, G.: Lightweight data indexing and compression in external memory. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 697–710. Springer, Heidelberg (2010)

4. Ferragina, P., Manzini, G.: On compressing the textual web. In: Proc. 3rd ACM International Conference on Web Search and Data Mining, pp. 391–400. ACM, New York (2010)
5. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representations of sequences and full-text indexes. ACM Trans. Algorithms 3, Article 20 (2007)
6. Golynski, A., Munro, J.I., Rao, S.S.: Rank/select operations on large alphabets: a tool for text indexing. In: Proc. 17th ACM-SIAM Symposium on Discrete Algorithms, pp. 368–373. ACM, New York (2006)
7. Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: Proc. 14th ACM-SIAM Symposium on Discrete Algorithms, pp. 841–850. SIAM, Philadelphia (2003)
8. Grossi, R., Gupta, A., Vitter, J.S.: When indexing equals compression: experiments with compressing suffix arrays and applications. In: Proc. 15th ACM-SIAM Symposium on Discrete Algorithms, pp. 636–645. SIAM, Philadelphia (2004)
9. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the I.R.E. 40, 1098–1101 (1952)
10. Kärkkäinen, J., Rantala, T.: Engineering radix sort for strings. In: Amir, A., Turpin, A., Moffat, A. (eds.) SPIRE 2008. LNCS, vol. 5280, pp. 3–14. Springer, Heidelberg (2008)
11. Kärkkäinen, J.: Fast BWT in small space by blockwise suffix sorting. Theoretical Computer Science 387, 249–257 (2007)
12. Katona, G.O.H., Nemetz, T.O.H.: Huffman codes and self-information. IEEE Transactions on Information Theory IT-22, 337–340 (1976)
13. Lauther, U., Lukovszki, T.: Space efficient algorithms for the Burrows-Wheeler backtransformation. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 293–304. Springer, Heidelberg (2005)
14. Mäkinen, V., Navarro, G.: Implicit compression boosting with applications to self-indexing. In: Ziviani, N., Baeza-Yates, R. (eds.) SPIRE 2007. LNCS, vol. 4726, pp. 229–241. Springer, Heidelberg (2007)
15. Manzini, G.: An analysis of the Burrows-Wheeler transform. Journal of the ACM 48, 407–430 (2001)
16. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Computing Surveys 39, Article 2 (2007)
17. Puglisi, S.J., Smyth, W.F., Turpin, A.: A taxonomy of suffix array construction algorithms. ACM Computing Surveys 39, 1–31 (2007)
18. Schwartz, E.S., Kallick, B.: Generating a canonical prefix encoding. Communications of the ACM 7, 166–169 (1964)
19. Seward, J.: Space-time tradeoffs in the inverse B-W transform. In: Storer, J., Cohn, M. (eds.) Proc. IEEE Data Compression Conference, pp. 439–448. IEEE Computer Society, Los Alamitos (2001)
20. Turpin, A., Moffat, A.: Housekeeping for prefix coding. IEEE Transactions on Communications 48, 622–628 (2000)
21. Vigna, S.: Broadword implementation of rank/select queries. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 154–168. Springer, Heidelberg (2008)

Median Trajectories*

Kevin Buchin[1], Maike Buchin[2], Marc van Kreveld[2], Maarten Löffler[3], Rodrigo I. Silveira[4], Carola Wenk[5], and Lionov Wiratma[2]

[1] Dept. of Mathematics and Computer Science, TU Eindhoven
k.a.buchin@tue.nl
[2] Dept. of Information and Computing Sciences, Utrecht University
{maike,marc}@cs.uu.nl, lionov@gmail.com
[3] Dept. of Computer Science, University of California, Irvine
mloffler@uci.edu
[4] Dept. de Matemàtica Aplicada II, Universitat Politècnica de Catalunya
rodrigo.silveira@upc.edu
[5] Dept. of Computer Science, University of Texas at San Antonio
carola@cs.utsa.edu

Abstract. We investigate the concept of a *median* among a set of trajectories. We establish criteria that a "median trajectory" should meet, and present two different methods to construct a median for a set of input trajectories. The first method is very simple, while the second method is more complicated and uses homotopy with respect to sufficiently large faces in the arrangement formed by the trajectories. We give algorithms for both methods, analyze the worst-case running time, and show that under certain assumptions both methods can be implemented efficiently. We empirically compare the output of both methods on randomly generated trajectories, and analyze whether the two methods yield medians that are according to our intuition. Our results suggest that the second method, using homotopy, performs considerably better.

1 Introduction

A relatively new type of geometric data that is being collected and analyzed more and more often is the *trajectory*: a path through space and time that a certain object traverses. This is due to technological advances like GPS, RFID tags, and mobile phones, and has caused an increase in demand for analysis possibilities. New analysis methods for trajectory data have been developed in the last few years, but a number of basic concepts are still lacking a satisfactory study. One of these concepts is the *median trajectory* for a given collection of

* This research has been supported by the Netherlands Organisation for Scientific Research (NWO) under BRICKS/FOCUS grant number 642.065.503, under the project GOGO, and under project no. 639.022.707. M. B. is supported by the German Research Foundation (DFG) under grant number BU 2419/1-1. M. L. is further supported by the U.S. Office of Naval Research under grant N00014-08-1-1015. R. I. S. is also supported by the Netherlands Organisation for Scientific Research (NWO). C. W. is supported by the National Science Foundation grant NSF CCF-0643597.

trajectories. Intuitively, a median trajectory is a trajectory that uses pieces of the trajectories of the collection and is somehow in the middle. However, it is not clear how this concept should be defined. In this paper we establish criteria that we believe a median trajectory should meet, and we develop two median definitions that meet these criteria. Furthermore, we give algorithms to compute the median trajectory according to these definitions, and analyze experimentally whether our definitions give useful output.

Trajectories. Trajectories are a type of geographic data that have a temporal and a spatial component. Trajectories describe the locations over time of an entity that can move. The entity can be a person, animal, vehicle, hurricane (eye of), shopping basket (with an RFID tag), or any other moving object. We assume that the movement is continuous, but is measured at a discrete set of times.

Formally, a trajectory is the time-stamped path taken by a moving object, and is typically represented by a sequence of tuples of points and time stamps, that are points in space-time, where space is two- or three-dimensional. A collection of m trajectories $\tau_1, \ldots, \tau_m$ therefore gives rise to an input size of $\Theta(nm)$. In some applications, the time stamps of the m trajectories are exactly the same, while in other applications they are different. In general, trajectories can be collected with different or irregular sampling rates, at different times, and data can be missing. In between time stamps, we have no knowledge of the movement of the entity. The standard assumption is that the moving object moves with constant velocity from a time-stamped point to the next time-stamped point. Therefore, the path of a trajectory is a polygonal curve that can self-intersect, and can have repeated vertices if the entity does not move. Often, the number of vertices of a trajectory is much larger than the number of trajectories, that is, $n \gg m$.

Trajectory analysis. Analysis methods for trajectories have been developed in GIScience and in data mining. Sets of trajectories can be analyzed in a variety of ways. They can be clustered into a collection of subsets that have a high within-subset similarity and a low across-subset similarity (e.g. [11,18] and many more). They can be classified if a clustering is given [19]. Movement patterns on them can be computed [5,12,17]. Movement patterns that have been defined and for which algorithms have been suggested are flocking, convoys, herds, leadership, commuting, encounter, and various others. These intuitively represent similar movement in a group (same location), similar movement over a time span (same heading), or movement to the same position (same destination).

Several analysis tasks require a definition of (and algorithms for) similarity of trajectories. For instance, a simple similarity measure for trajectories is the average distance at corresponding times. With a similarity measure, or its inverse, a distance measure, clustering methods can easily be given. Single linkage and complete linkage clustering need a similarity measure only, and that defines the clustering. On the other hand, k-means and k-medoids clustering requires a definition of the mean and the median, respectively, regardless of whether the data are numbers or trajectories.

Mean and median trajectories. The intuition behind a mean trajectory is that it averages locations, one of each trajectory, like a center of gravity. In contrast, the intuition behind a median trajectory is that it always is central with respect to the number of given trajectories. Imagine a collection of GPS tracks from hikes by different people on different days. The hikers may have followed the same route globally, but there may have been options like going left around a lake or right, or taking a detour to a viewpoint. From their tracks, we want to extract a good global route. Notice that if in such a data set, seven hikers went left around a lake and three went right, then a mean trajectory would actually go through the lake, whereas a median trajectory would go with the group of seven. Similarly, with a side path to a viewpoint and back, if the majority goes to the viewpoint, then a median trajectory should do so as well. A mean trajectory might go partially to the viewpoint, which does not make sense in this context.

Overview of results. In Section 2 we discuss the idea of median trajectories. No definition has been suggested yet, so we investigate properties that a suitable median should have. We first propose a simple definition (*simple median*) that directly follows the definition of a *level* in an *arrangement of lines.* We also propose a more refined definition (*homotopic median*) that uses geometric and topological concepts, and may be better suited to most applications that involve trajectories. Then we discuss the maximum combinatorial complexity of a median according to these definitions.

In Section 3 we present algorithms that compute median trajectories according to the two definitions. We can compute the simple median in $O((nm)^2)$ time, and the homotopic median in $O((nm)^{2+\epsilon})$ time for any $\epsilon > 0$, in the worst case. Here, m is the number of given trajectories and n is the maximum complexity of any trajectory. We improve our algorithms for practical situations. We can compute the simple median in $O((nm+k)\alpha(nm)\log(nm))$ time, where α is the inverse Ackermann function and k is the number of vertices of the median, i.e., the output complexity. Under certain assumptions related to the sampling of the trajectory, we can compute the homotopic median in $O(nm\log^2(nm) + (nm + k)\alpha(nm)\log(nm))$ time. We note that $k = O((nm)^2)$ in the worst case, as we show in Section 2.2. One would expect that typically, $k = \Omega(n)$ and $k = O(nm)$.

In Section 4 we give results of tests that we obtain from an implementation. They mainly serve to analyze the quality of the median trajectories. In Section 5 we discuss our results and suggest directions of further research.

2 On the Definition of a Median Trajectory

The *mean* and the *median* are notions that define a "middle" of a set of data of a certain type. For sets of numbers, their definitions are clear, but for points in the plane, for instance, there are already many different possibilities. Existing notions of middle include the center of gravity, the center of the smallest enclosing disk (also known as *Gaussian center* or *Steiner center* [9,10]), the point that minimizes the sum of distances to the other points (the *Weber point* [8]), or the *center point* [4]. The one-dimensional equivalents of these notions correspond to the mean or the median. Observe that for some point sets, such as a set of points in convex position, no point in the set would intuitively be the mean or median. For a set of m real-valued, continuous functions the median of the functions corresponds (assuming m is odd) to the $\lceil m/2 \rceil$-*level* in the arrangement of the graphs of the functions. Figure 1 (left) shows an example for lines.

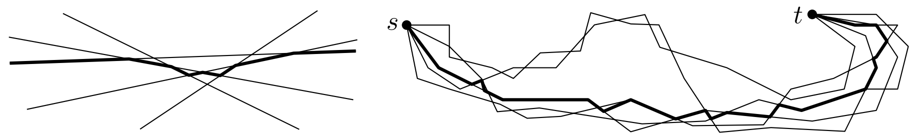


Fig. 1. Left, the median level in an arrangement of lines. Right, a median trajectory from s to t that always switches trajectory at every intersection.

Let us consider median trajectories. Trajectories include a temporal component as well as a spatial component, but it is not clear whether a median can take the temporal component into account in a useful way. We discuss some examples. Suppose the trajectories came from a group of animals that were traveling in a herd. Then we can use the temporal component because we know that the animals were together at any point in time. Next, suppose that the animals were traveling solitary, according to a similar route. Then they traveled on different days or months, and we cannot use the temporal component. Even if the animals had the same starting location of the route, we cannot simply align the starting times of the travel, because one animal may have been held up due to a predator, which upsets the time correspondence that we assumed at the start. The same is true for trajectories of cars with the same origin and destination: an initial time correspondence may easily be upset due to traffic lights or traffic conditions.

Thus, in many situations we want a median trajectory that does not take the temporal component into account. Similar motivation was given for trajectory similarity measures: many of these are partly shape-based, like dynamic time warping or largest common subsequence, or fully shape-based, like Hausdorff distance or Fréchet distance. Hence, we will concentrate on medians of trajectories based on the path of the trajectory. The median that we will define and compute will therefore be the path of a median trajectory. With slight abuse of terminology, we will just write "median trajectory" for brevity. We note that with a temporal component, some research on modeling motion and kinetic data structures is related to the median (or mean) trajectory (e.g. [1,2,3]).

2.1 Requirements for a Median Trajectory

Let a set $T = \{\tau_1, \ldots, \tau_m\}$ of m trajectories be given, each containing n vertices. We assume for convenience that all trajectories start at the same point s and end at the same point t; this is a strong and unrealistic assumption but we are interested in a clean definition of the median where behavior at the ends is not considered important. We also assume that no trajectory passes through s or t a second time and that s and t are incident to the unbounded face of the arrangement of curves corresponding to the trajectories. Note that we also assume a direction on the trajectories, namely from s to t, and for convenience we assume that m is odd. Finally, we assume that the curves do not touch or coincide with each other or themselves at any point, unless they cross, and no three trajectories pass through a common point. Several of these assumptions can be removed, but they make the description easier.

We list several required properties of the median trajectory:

1. The median trajectory is a polygonal curve from s to t.
2. Any point on the median trajectory lies on some trajectory of the input.
3. For any point p on the median trajectory, the minimum number of distinct trajectories that p must cross to reach the unbounded face (including the one(s) on which p lies) is $(m+1)/2$.

Besides these requirements, a number of desirable properties of the median trajectory can be given: Its length, total angular change, and number of vertices should be about the same as in the input trajectories. Finally, the median trajectory should be robust with respect to outliers: if ten trajectories follow the same route but one or two are completely different, the presence of these two outliers should not influence the median trajectory much. In particular, in the presence of outlier trajectories, the third property should be restated with m the number of non-outlying trajectories instead of the total number of trajectories.

Let $\mathcal{A}$ be the arrangement formed by the (paths of the) trajectories in T. It is composed of $O(nm)$ line segments and therefore it may have complexity up to $\Theta((nm)^2)$. The median trajectory is a path that follows edges of this arrangement. In the immediate neighborhood of s, it is clear how the median trajectory leaves s. Since we assume that s is on the outer face, we can order the m edges adjacent to s with the first and last edge adjacent to the outer face. Then the edge the median starts on is simply the $(m/2)$nd edge in the order.

Simple definition. Inspired by the median level in an arrangement of lines, we can give a simple definition of the median: It is the trajectory obtained after leaving s in the only possible way, and then switching the trajectory at every intersection point, following the next trajectory in the forward direction (see Figure 1 (right)). Note that if the trajectories are x-monotone, then this definition gives the same result as the $\lceil m/2 \rceil$-level or median function given before. We refer to a median by this definition as a *simple median*. The proofs of the following and other lemmas are omitted due to space constraints.

Lemma 1. *The simple median satisfies the three required properties for a median.*

Although this definition often gives desired results, it can behave badly when trajectories are self-intersecting, see Figure 2 (left). All three trajectories make the loop, but the median does not. Similarly, when the trajectories are not self-intersecting, the majority route can also be missed. In Figure 2 (middle), two

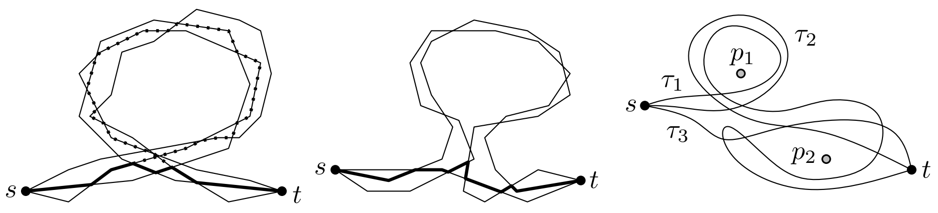


Fig. 2. Left, a loop in all trajectories makes the outcome (bold) of always switching trajectory undesirable. Middle, similar undesirable outcome can occur when no trajectory has self-intersections. Right, homotopy with respect to two poles.

trajectories go up and make a detour, while one trajectory stays low. The simple median in this situation also stays low.

Homotopy definition. To deal with this situation better, we identify parts of the plane with respect to which the median should behave the same as most of the input trajectories. In both examples of Figure 2, we have a region in the plane that is a bounded face of the arrangement $\mathcal{A}$ that is relatively large. We propose placing *poles* in such large faces and require that the median goes in the same way around the poles as the input trajectories, using the concept of *homotopy.*

We make this more precise. Let $T = \{\tau_1, \ldots, \tau_m\}$ be the input trajectories and let $P = \{p_1, \ldots, p_h\}$ be a set of h poles which are assumed to not lie on any trajectory. Since the trajectories all go from s to t, we can use deformability of the trajectories into each other in the punctured plane [15]. Two trajectories τ_i and τ_j are *homotopic* if one can be deformed continuously into the other without passing over any pole, and while keeping s and t fixed. In Figure 2 (right), τ_1 and τ_2 are homotopic to each other, while τ_3 is not homotopic to τ_1 or τ_2.

We first discuss how we find the median when all trajectories in T are homotopic with respect to P. We now use a variation of the trajectory switching approach: follow the median over the correct edge at s. Assume we have followed the median and we are on an edge of trajectory τ_i and we encounter an intersection v with a trajectory τ_j. If the median so far, concatenated with trajectory τ_j from v until t, has the same homotopy type as the input trajectories, then we switch to τ_j, otherwise we ignore the intersection and stay on τ_i. This approach maintains the invariant that if we would simply stay on the current trajectory until t, the homotopy type of the median is correct. A median by this definition is referred to as a *homotopic median.* In Figure 2 (left) the dotted loop would be included in the homotopic median.

Lemma 2. *The homotopic median satisfies the required properties for a median.*

The remaining question is how to find a set of poles P such that all trajectories in T are homotopic with respect to it. A number of different strategies for this are conceivable. We choose to use a simple approach that places a pole in a face of $\mathcal{A}$ whenever it is larger than r, i.e., a disk of size r fits in the face, for some value of r to be determined later, motivated by the fact that large faces are more likely to be important. This choice gives no guarantee that the trajectories will be homotopic though. To solve this we could either increase r or allow some of the trajectories to have a different homotopy, and instead compute the median for a subset $T' \subset T$, i.e., effectively treating the remaining trajectories as outliers.

2.2 The Complexity of Median Trajectories

Since the arrangement $\mathcal{A}$ formed by the m trajectories has complexity $O((nm)^2)$, the median trajectory cannot have more edges than that. In the full version of the paper we show that m non-self-intersecting trajectories with n edges each can indeed give rise to a median of complexity $\Omega((nm)^2)$, using both definitions. We also note that using the median level lower bound for arrangements of lines, we immediately obtain an $\Omega(nm \log m)$ lower bound on the complexity of the median of x-monotone trajectories.

3 Algorithms to Compute a Median Trajectory

In this section we show that both methods can be implemented efficiently. The simple median can be computed in $O((nm)^2)$ time in the worst case, while the homotopic median can be computed in $O((nm)^2 \log(nm))$ time in the worst case. In practice, running times will be faster because they depend on complexities of intermediate results that should be much less than the worst-case situations. In particular, let A be the complexity of the arrangement formed by the nm edges of the trajectories. Although $A = O((nm)^2)$ and this is tight in the worst case, for trajectories we typically expect it to be much smaller. Therefore, making the time bound depend on A instead of $(nm)^2$ is desirable. Similarly, let h be the number of poles. Then $h = O(A) = O((nm)^2)$, but only large enough faces give poles so h is typically much smaller than A. Finally, the complexity of the median itself, the output size k, is $O(A) = O((nm)^2)$, but typically we expect it to be smaller. Notice that h or k can be large when the other is small.

3.1 Computing the Simple Median

A simple algorithm to compute the median with the simple definition is via the construction of the arrangement $\mathcal{A}$. The arrangement can be constructed in $O(nm \log(nm) + A)$ time, where A is the complexity of the arrangement [13]. Then we simply follow the median trajectory through this arrangement, taking $O(1)$ time at every intersection point or trajectory vertex. Hence, this algorithm takes $O(nm \log(nm) + A) = O((nm)^2)$ time.

For an output-sensitive algorithm, we use Har-Peled's randomized algorithm for an on-line walk in a planar arrangement [14]. This algorithm has an expected runtime of $O((nm+I)\alpha(nm+I)\log(nm))$ in an arrangement of nm line segments, where I is the number of intersections between the walk and the arrangement, and α denotes the inverse Ackermann function. In our case, $I = k$ is the complexity of the median, because the median switches trajectories at every intersection. Furthermore, $I = O((nm)^2)$ and then $\alpha(nm + I) = O(\alpha(nm))$.

Theorem 1. *The simple median of m trajectories with n edges can be computed in $O((nm)^2)$ time or in $O((nm + k)\alpha(nm)\log(nm))$ expected time, where k is the size of the output.*

3.2 Computing the Homotopic Median

We give an algorithm that computes the homotopic median for a given value of r. If for the resulting poles not all trajectories are homotopic, we compute the median trajectory of the largest homotopy class with respect to these poles. The algorithm consists of three steps.

1. Compute poles, one for each face in which a disk of radius r fits.
2. Compute the homotopy type of each trajectory, and determine the type that occurs most often. Remove all trajectories that do not have this type.
3. Follow the median from s: at every intersection, determine whether the continuation on the new trajectory yields the correct homotopy type (when the new trajectory is followed to t).

Step 1 can be performed in $O(nm\log(nm) + A)$ time by constructing the arrangement [13] and then computing the medial axis in each face [7]; the medial axis gives the largest disk that fits inside. Step 2 can be performed using an algorithm of Cabello et al. [6]: Deciding whether two paths are homotopic takes $O(n\sqrt{h}\log h)$ time, assuming the two paths have n edges and there are h poles. The algorithm makes use of a spanning tree on the poles so that any line intersects at most $O(\sqrt{h})$ edges of this spanning tree, which can be constructed in $O(h^{1+\epsilon})$ time for any $\epsilon > 0$. We can adapt this algorithm to determine the largest subset of homotopic trajectories. Step 3 can be performed by explicitly computing the arrangement of the trajectories. We trace the median, switching between two intersecting trajectories only if this intersection also occurs in the universal cover. The running time of this step is $O(nm\log(nm) + A)$ time to compute the arrangement, and $O(mn\sqrt{h} + A)$ time to trace the median.

Theorem 2. *The homotopic median of m trajectories with n edges can be computed in $O((nm)^{2+\epsilon})$ time or in $O((nm\sqrt{h} + k)\alpha(nm)\log(nm) + h^{1+\epsilon} + A)$ expected time for any $\epsilon > 0$, where h is the number of poles, A is the arrangement size and k is the output size.*

Sampling assumption. It seems reasonable to assume that the size of faces that are relevant and get a pole, represented by r, is not much less than the length s of the longest edge in any trajectory. Suppose for instance that $r \leq s/6$. Then the trajectories are sampled so sparsely that it can happen that two trajectories have exactly the same edge of length $2r$, whereas one traversed in that time unit a distance $6r$ using three sides of a square of side length $2r$ instead of one side; note that such a square contains a disk of radius r and therefore would contain a pole. However, we could not know that the trajectories were very different, although the choice of r suggests that this is relevant. This makes the assumption $r = \Omega(s)$ reasonable; we refer to it as the *sampling assumption.* Under this assumption, we can improve the running times of our algorithms.

For Step 1, we determine all faces that get a pole without constructing the arrangement. Let D be a disk of radius r centered at the origin. Take the Minkowski sum of every edge of every trajectory and D, and compute the union of these $O(nm)$ “race tracks”. The complement of the union contains parts of all faces that are large enough, although the same face of $\mathcal{A}$ may appear as several faces in the complement of the union. Notice that homotopic equivalence is not influenced if any face of $\mathcal{A}$ has more than one pole. The sampling assumption implies that all race tracks are *fat objects* of similar size, and hence the union complexity is bounded by $O(nm)$ [20]. We use the algorithm of Kedem et al. [16] to construct it in $O(nm \log^2(nm))$ time. This gives us a set of $h = O(nm)$ poles.

For efficiency reasons in Steps 2 and 3, we must avoid having two poles closer than $2r$. Two such poles would lie in the same face of $\mathcal{A}$, so we can remove either one. We determine a subset of the poles such that every big face of $\mathcal{A}$ has at least one pole in the subset, and any two poles in the subset are at least $2r$ apart. We do this by computing the Delaunay triangulation of the poles. We then remove all edges that have length more than $2r$, and choose one pole per connected component in our subset. Let P be this subset of poles. The idea is to construct a spanning tree on P with stabbing number $O(1)$ for line segments of length at most s.

We do this as follows: take a set of vertical lines that are exactly r apart. Within each vertical slab, connect the poles by y-coordinate. Across vertical slabs, consider every two consecutive non-empty slabs; there may be empty slabs in between. We connect the rightmost point of the left slab with the leftmost point of the right slab. Any segment of length at most s crosses $O(1)$ slabs, and due to vertical spacing within a slab, it intersects $O(1)$ spanning tree edges per slab. Hence, the spanning tree on P has $O(1)$ stabbing number for line segments of length at most s, in particular, for the edges of the trajectories.

Now we again use the same algorithm as above, but replace the spanning tree with stabbing number $O(\sqrt{h})$ by this spanning tree with $O(1)$ stabbing number. The resulting expected running time for Step 3 is $O((nm + k)\alpha(nm)\log(nm))$.

Theorem 3. *The homotopic median of m trajectories with n edges can be computed in $O(nm \log^2(nm) + (nm + k)\alpha(nm)\log(nm))$ expected time, where k is the size of the output, if the sampling assumption is satisfied.*

4 Experimental Results for Median Trajectories

In this section we present experimental results that aim at analyzing the quality of the medians generated by our two definitions. The experiments compare the definitions quantitatively, with respect to the desirable properties mentioned in Section 2—number of vertices, total length, and total turning angle—and also qualitatively, by analyzing visually in which cases one or the other method produces counterintuitive results.

Experimental set-up. We implemented a random trajectory generator that generates sets of “similar trajectories”. For each of these sets of trajectories, the medians for both definitions were computed and analyzed.

The random trajectory generator starts by generating a given number of waypoints uniformly distributed at random inside a rectangle, with the restriction that two consecutive waypoints are further than some minimum distance apart, and for three consecutive waypoints, the angle between them is not too small (to avoid U-turns). Given these waypoints, a given number of trajectories is generated that all follow the sequence of waypoints. For each trajectory, edges are generated that have a variation in length and in heading, but always somewhat towards the next waypoint. A waypoint is considered reached if a trajectory has a vertex within a certain distance from it, and then the next edge proceeds towards the next waypoint. The longest-to-shortest edge length ratio is 2, and the heading is up to 45° off from being directed to the next waypoint. With some small probability, a trajectory may temporarily not head to the next waypoint but somewhere else, resulting in outliers. Also with a small probability, a generated trajectory may skip waypoints. All trajectories start at the first waypoint and end at the last one.

Data set. We generated many sets of 8 waypoints and then 9 trajectories for each of them. A set was accepted if at least 6 out of 9 trajectories were homotopic for a fixed value of r. For each accepted set, the medians according to both definitions were computed, giving the length, angular change, and number of vertices for both. We distinguished four types of waypoint sets, depending on two properties of the polygonal line implied by the sequence of 8 waypoints: we compare no self-intersections (1) to self-intersections (2), and low angular change (a) to high angular change (b) (angles below or above 3.8π). We repeatedly generated sets until we had 100 sets in each of the four classes. Note that the classes refer to the properties of the waypoints, not the trajectories themselves (trajectories may self-intersect even if the waypoint sequence does not).

Table 1. Left: average length and standard deviation according to both definitions (μ_S and σ_S for the simple median; μ_H and σ_H for the homotopic median), where the average length of the input trajectories is normalized to 1. Middle: same for angular change. Right: same for number of vertices, again after normalization.

	length				angular change				no. of vertices			
	μ_S	σ_S	μ_H	σ_H	μ_S	σ_S	μ_H	σ_H	μ_S	σ_S	μ_H	σ_H
1a	0.960	0.171	0.986	0.060	5.387	1.025	4.680	0.747	3.446	0.690	3.188	0.476
1b	0.947	0.186	0.992	0.060	5.757	1.148	4.977	0.900	3.571	0.776	3.254	0.554
2a	0.513	0.263	0.963	0.115	3.807	2.112	4.656	0.970	2.071	1.062	3.433	0.753
2b	0.520	0.268	0.956	0.087	4.120	2.402	4.698	1.009	2.143	1.160	3.426	0.720

Results. We summarize the results in Table 1. Observe that with self-intersections the simple median method gives medians that are on average significantly shorter than the input trajectories. The other tables also indicate that often the simple method does not deal well with self-intersecting trajectories. Regarding the angular change and the number of vertices, we see that the median has much higher values than the average input. The fact that the number of vertices of the median is higher is not surprising, and this nearly implies that the angular change is higher as well. This higher angular change is undesirable, because it can make the median trajectory appear very different from the input trajectories. Comparing the definitions, the lower standard deviation suggests that the results for the homotopic median are more consistent.

Visual inspection showed that the simple median occasionally made "errors" (against intuition) even for inputs without self-intersections, and nearly always for inputs with self-intersections. The homotopic median nearly always gave intuitive results, although an occasional "error" could be observed, due to the absence of poles in regions that are split by unrelated pieces of the trajectories. We also tested how the number of vertices of the median is influenced by the number of trajectories. There seems to be a linear dependence, suggesting that the output size $k = \Theta(mn)$, but this observation is highly dependent on our random trajectory generator. In summary, except for the high angular change and occasional missing parts, the homotopic median performs well even for intersecting trajectories.

5 Discussion and Future Research

We discussed the fundamental—but up to now missing—concept of the median of a set of trajectories. We make a first step in this direction by proposing necessary and desirable conditions that a trajectory median should satisfy. Based on them, we presented two definitions of the path of a median trajectory of a set of trajectories, together with efficient methods to compute them. We also proved properties of the resulting medians and analyzed them experimentally.

Given the importance of the concept of a median trajectory and its novelty, we believe this paper opens up many venues of further research. We made several restrictions in this paper that may be unrealistic. We assumed the start and end points of all trajectories to coincide and to lie in the unbounded face. We also assumed that most trajectories are similar enough; the homotopy method can deal with some trajectories that are outliers, but it cannot deal properly with the situation where *parts* of many trajectories are outliers. This can result in a situation where the largest homotopy class has only one trajectory, whereas an intuitively correct median may still exist. We also assumed the parameter r to be given; it would be desirable to choose it automatically in an efficient manner.

We have not addressed the question how to assign time stamps to the median or how to use the time stamps of the input to guide the computation. And of course it may be possible to define a median that has good properties in a completely different way. Finally, it would be interesting to test the definitions of medians for various types of real-world data, instead of generated data.

References

1. Agarwal, P.K., de Berg, M., Gao, J., Guibas, L.J.: Staying in the middle: Exact and approximate medians in R1 and R2 for moving points. In: Proc. CCCG, pp. 43–46 (2005)

2. Agarwal, P.K., Gao, J., Guibas, L.J.: Kinetic medians and kd-trees. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 5–16. Springer, Heidelberg (2002)
3. Agarwal, P.K., Guibas, L.J., Hershberger, J., Veach, E.: Maintaining the extent of a moving point set. Discrete Comput. Geom. 26, 353–374 (2001)
4. Amenta, N., Bern, M.W., Eppstein, D., Teng, S.-H.: Regression depth and center points. Discrete Comput. Geom. 23, 305–323 (2000)
5. Buchin, K., Buchin, M., Gudmundsson, J., Löffler, M., Luo, J.: Detecting commuting patterns by clustering subtrajectories. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 644–655. Springer, Heidelberg (2008)
6. Cabello, S., Liu, Y., Mantler, A., Snoeyink, J.: Testing homotopy for paths in the plane. Discrete Comput. Geom. 31, 61–81 (2004)
7. Chin, F.Y.L., Snoeyink, J., Wang, C.A.: Finding the medial axis of a simple polygon in linear time. Discrete Comput. Geom. 21, 405–420 (1999)
8. Durocher, S., Kirkpatrick, D.: The projection median of a set of points. Comput. Geom. 42, 364–375 (2009)
9. Durocher, S., Kirkpatrick, D.G.: The Steiner centre of a set of points: Stability, eccentricity, and applications to mobile facility location. Int. J. Comput. Geom. Appl. 16, 345–372 (2006)
10. Durocher, S., Kirkpatrick, D.G.: Bounded-velocity approximation of mobile Euclidean 2-centres. Int. J. Comput. Geom. Appl. 18, 161–183 (2008)
11. Gaffney, S., Smyth, P.: Trajectory clustering with mixtures of regression models. In: Proc. 5th KDD, pp. 63–72 (1999)
12. Gudmundsson, J., van Kreveld, M., Speckmann, B.: Efficient detection of patterns in 2D trajectories of moving points. GeoInformatica 11, 195–215 (2007)
13. Halperin, D.: Arrangements. In: Goodmann, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Comput. Geom, pp. 529–562. Chapman & Hall/CRC, Boca Raton (2004)
14. Har-Peled, S.: Taking a walk in a planar arrangement. SIAM J. Comput. 30(4), 1341–1367 (2000)
15. Hershberger, J., Snoeyink, J.: Computing minimum length paths of a given homotopy class. Comput. Geom. 4, 63–97 (1994)
16. Kedem, K., Livne, R., Pach, J., Sharir, M.: On the union of jordan regions and collision-free translational motion amidst polygonal obstacles. Discrete Comput. Geom. 1, 59–70 (1986)
17. Laube, P., Purves, R.S.: An approach to evaluating motion pattern detection techniques in spatio-temporal data. Comp., Env. and Urb. Syst. 30, 347–374 (2006)
18. Lee, J., Han, J., Whang, K.-Y.: Trajectory clustering: a partition-and-group framework. In: Proc. ACM SIGMOD Int. Conf. Man. of Data, pp. 593–604 (2007)
19. Lee, J.-G., Han, J., Li, X., Gonzalez, H.: TraClass: Trajectory classification using hierarchical region-based and trajectory-based clustering. In: PVLDB 2008, pp. 1081–1094 (2008)
20. van der Stappen, A.F., Halperin, D., Overmars, M.H.: The complexity of the free space for a robot moving amidst fat obstacles. Comput. Geom. 3, 353–373 (1993)

Optimal Cover of Points by Disks in a Simple Polygon

Haim Kaplan[⋆], Matthew J. Katz[⋆⋆], Gila Morgenstern[⋆⋆⋆], and Micha Sharir[†]

Abstract. Let P be a simple polygon, and let Q be a set of points in P. We present an almost-linear time algorithm for computing a minimum cover of Q by disks that are contained in P. We generalize the algorithm above, so that it can compute a minimum cover of Q by homothets of any fixed compact convex set $\mathcal{O}$ of constant description complexity that are contained in P. This improves previous results of Katz and Morgenstern [20]. We also consider the disk-cover problem when Q is contained in a (not too wide) annulus, and present a nearly linear algorithm for this case too.

1 Introduction

Let P be a simple n-gon in the plane, and let Q be a set of m points in P. A *disk cover of Q with respect to P* is a set $\mathcal{D}$ of disks (of variable radii), such that the union of the disks of $\mathcal{D}$ covers (i.e., contains) Q and is contained in P. In other words, each disk $D \in \mathcal{D}$ is contained in P, and each point $q \in Q$, lies in at least one disk $D \in \mathcal{D}$. A *minimum disk cover of Q with respect to P* is a disk cover of Q with respect to P of minimum cardinality. The problem of computing a minimum disk cover of Q with respect to P was introduced and studied by Katz and Morgenstern [20]. They also considered the case where the

[⋆] School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: haimk@post.tau.ac.il. Work by Haim Kaplan was partially supported by the U.S.-Israeli Binational Science Foundation, project number 2006-204, and by grant 975/06 from the Israel Science Fund.

[⋆⋆] Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel. E-mail: matya@cs.bgu.ac.il. Work by Matthew Katz was partially supported by the MAGNET program of the Israel Ministry of Industry, Trade & Labor (CORNET consortium), and by the Lynn and William Frankel Center for Computer Sciences.

[⋆⋆⋆] Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel. E-mail: gilamor@cs.bgu.ac.il. Work by Gila Morgenstern was partially supported by the Lynn and William Frankel Center for Computer Sciences.

[†] School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA. E-mail: michas@post.tau.ac.il. Work by Micha Sharir was partially supported by NSF Grant CCF-08-30272, by grant 2006-194 from the U.S.-Israeli Binational Science Foundation, by grant 338/09 from the Israel Science Fund, Israeli Academy of Sciences, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

covering objects are homothets (contained in P) of any fixed compact convex set $\mathcal{O}$ of constant description complexity. In both cases, exact polynomial-time solutions were presented. In this paper we present alternative and significantly faster solutions for both disks and homothets, and also consider the case where the points are in an annulus. All our solutions run in close to linear time.

Background. Geometric covering problems have been studied extensively. These problems are instances induced by geometric settings of the well-known set cover problem. Most of these instances are known to be NP-hard. Let us briefly review several geometric covering problems that are related to the problems studied in this paper.

In the unit disk cover problem, the goal is to cover a given set of points with the smallest possible number of unit disks. A polynomial-time approximation scheme (PTAS) for this problem was given by Hochbaum and Maass [17]. In the discrete version of this problem, the covering unit disks must be selected from a given set of unit disks. Until recently only constant-factor approximation algorithms were known for the discrete version; see [2,5,6,30]. This was recently improved by Mustafa and Ray [29], who presented a PTAS for the discrete version (as well as for several other problems), which is based on local search.

Hurtado et al. [19] studied the related problem of computing a minimum enclosing disk of a given set of m points, whose center must lie in a given convex n-gon; they presented an $O(m+n)$-time algorithm for this problem. The 2-center problem with obstacles was studied by Halperin et al. [16]. In this problem, the goal is to find two congruent disks of smallest radius whose union covers a given set of m points and whose centers lie outside a given set of disjoint simple polygons with a total of n edges. They presented a randomized $O(n \log^2(mn) + mn \log^2 m \log(mn))$ expected time algorithm for this problem. The analogous 1-center problem was studied by Halperin and Linhart [15], who presented an $O((m+n)\log(mn))$-time algorithm for this problem.

The solutions of Katz and Morgenstern [20] for the problems mentioned above are based on the "perfect graph approach", previously used in the solution of several art-gallery problems, under restricted models of visibility; see, e.g., [21,22,27,28,32]. In this approach, one first defines a graph G corresponding to the input scene. Next, the following two theorems are proven: (i) There is a one-to-one correspondence between a minimum cover of the desired kind (e.g., disk cover) and a minimum clique cover of G, and (ii) G is perfect. Note that the second claim is crucial, since, in general, minimum clique cover is NP-complete, but is polynomial for perfect graphs [13,14] and in particular for chordal graphs[1] [11]. The algorithms of Katz and Morgenstern consist of three stages. First, construct G, next, find a minimum clique cover of G, and finally, construct the cover of Q corresponding to the minimum clique cover of G. The bottlenecks of their algorithms are the first and last stages, which, in the case of covering by disks within a simple polygon, required $O(nm^2)$ time.

[1] A graph is *chordal* if every cycle of at least four edges has a *chord*, i.e., an edge connecting two non-consecutive vertices of the cycle; see [13].

In this paper we take a different approach, avoiding the explicit construction of the graph G, and computing the cover itself by following the algorithm of Gavril [11] for finding a minimum clique cover in a chordal graph, and by exploiting the special geometric structure of G. This leads to improved solutions, which, when carefully implemented, run in nearly linear time.

The rest of this paper is organized as follows: In Section 2, we describe an algorithm for computing a minimum cover of Q by disks contained in P in $O((n + m(\log n + \log^2 m)))$ time and $O(n + m)$ space. In Section 3, we extend this result to the case of $\mathcal{O}$-cover, that is, computing a minimum cover of Q by homothets of an object $\mathcal{O}$ which are contained in P, where $\mathcal{O}$ is as above. We show that such a cover can be computed within the same bounds. Finally, in Section 4 we consider the case where the point set Q is contained in a "not too wide" annulus R, and give an $O(m \log m)$-time algorithm for computing a minimum-disk cover of Q by disks contained in R.

2 Minimum Disk Cover in a Simple Polygon

Let P be a simple polygon with n edges and let Q be a set of m points inside P. We present a nearly linear time algorithm for finding a minimum cover of Q by disks contained in P.

Consider the Voronoi diagram of the relatively open edges and reflex vertices of P, confined to within P. We refer to these relatively open edges and reflex vertices collectively as *boundary features* (or simply *features*) of P, and let P^* denote the set of these features. The *medial axis* M is the network of vertices, straight edges, and parabolic arcs of this Voronoi diagram which are strictly inside P, including also the non-reflex vertices of P. More precisely, a vertex of M which is not a vertex of P is a point at equal and smallest distance from three features of P (including the case where two of these features are an edge e and a reflex endpoint of e). An edge of M is the locus of all points at equal (and smallest) distance from two features of P (this time excluding the case of an edge and one of its endpoints). It is well known (and easy to show) that the network M is in fact a tree; that is, it is a connected network without cycles. See Fig. 1(a).[2]

As we will argue shortly, when constructing a disk cover of Q inside P, it suffices to consider disks whose centers lie on M, and in fact only maximal such disks, whose boundary touches ∂P (necessarily in at least two points).

The proof of Lemma 1 below, a major geometric component of our analysis, can be found in the full version of this paper.[3]

Lemma 1. *For each point $q \in Q$, the portion M_q of the medial axis consisting of centers of maximal disks that contain q and are contained in P is connected.*

[2] The tree property may fail if we include in M edges at equal distance from an edge e of ∂P and from a reflex endpoint of e.

[3] The full version of this paper can be found at: `http:/www.cs.bgu.ac.il/~gilamor/papers/KKMS10.pdf`

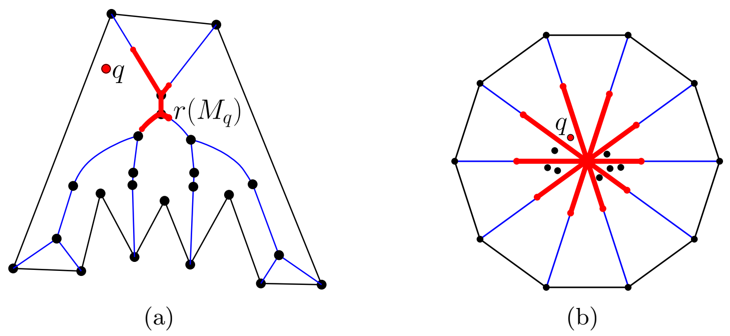


Fig. 1. (a) The medial axis of a simple polygon and the subtree M_q of a point $q \in Q$ (drawn bold, with its root $r(M_q)$ highlighted). (b) The complexity of each subtree M_q is $\Theta(n)$ in this example.

Consider the graph G whose vertices are the points of Q, and there is an edge between two points $p, q \in Q$ if there exists a disk containing both p and q and contained in P. Clearly there is such a disk if and only if there is a disk containing p and q whose center is on M and whose radius is the distance from its center to ∂P. This follows by noting that every disk D contained in P is contained in a maximal disk of the above kind, which is obtained by inflating D about its center until it touches ∂P, and then by moving its center away from the contact with ∂P, maintaining that contact, until a second contact is made.

For each $q \in Q$, the portion M_q of M considered in Lemma 1 is equal to the intersection of M with the Voronoi cell of q in the diagram of $P^* \cup \{q\}$. The lemma asserts that M_q is a connected subset of M. We refer to M_q as a *subtree* of M, but note that the leaves of M_q (points of M_q whose removal does not disconnect M_q) need not necessarily be vertices of M, but can lie in the relative interior of edges of M; see Fig. 1(a). It follows by definition that, by identifying each point q of Q with its subtree M_q, G becomes the intersection graph of these subtrees. Therefore, by the characterization of Buneman and Gavril [4,12], G is a *chordal* graph.

We note that computing the subtrees M_q explicitly is too expensive, because their overall complexity can be $\Theta(mn)$ in the worst case, as depicted in Fig. 1(b).

As shown by Katz and Morgenstern [20], a cover of Q by disks contained in P corresponds to a clique cover of G and vice versa. This also follows from the fact that subtrees of a tree T satisfy the 2-*Helly-property*: if every pair of subtrees in a given collection intersect, then they all have a common intersection [13]. Therefore, a clique of G can be covered by a single disk which is a maximal disk centered at a common point of all subtrees M_q corresponding to the points q of the clique. (The converse direction is trivial.) So our problem is to find a minimum clique cover of G. As is known [11], the chordality of G implies that this latter problem is solvable in polynomial time, but, exploiting the special geometric structure of the graph G, we are able to solve it particularly efficiently, by an algorithm which runs in $O(n+m(\log n+\log^2 m))$ time (improving upon the cubic algorithm of Katz and Morgenstern in [20] mentioned in the introduction).

We follow the algorithm of Gavril [11] for finding a minimum clique cover in a chordal graph. The algorithm is greedy and is based on the fact (see, e.g., [13]) that a chordal graph contains a *simplicial vertex* v, that is, a vertex whose neighbors induce a clique. The greedy clique cover algorithm takes as the first clique in the cover such a simplicial vertex v and its neighbors. It then deletes v and its neighbors and iterates this step on the subgraph induced by the remaining vertices (which is still chordal). It is not hard to see that the output clique cover is indeed minimum, because the simplicial vertices picked at each iteration form an independent set in the original G, and clearly the size of any clique cover is at least the size of this (or any) independent set.

We can implement this algorithm using the subtree representation of G, as follows. Root M at an arbitrary vertex of ∂P. Making M rooted induces a root for each subtree M_q, for $q \in Q$, which we denote by $r(M_q)$. Note that $r(M_q)$ is not necessarily a vertex of M but can lie in the relative interior of an edge. See Fig. 1(a).

Pick M_q such that the (appropriately defined) subtree of M rooted at $r(M_q)$ contains no other root. The points of Q corresponding to all subtrees which contain $r(M_q)$ are the first clique in our cover (q is a simplicial vertex). The disk corresponding to this clique can be taken to be the maximal disk within P centered at $r(M_q)$. We then delete all subtrees containing $r(M_q)$ and iterate, until all of Q is exhausted.

We now present an efficient implementation of this algorithm. It consists of the following steps.

1. For each point $q \in Q$ compute an "anchor point" $A(q) \in M_q$.
2. Starting from each $A(q)$, search M to find the corresponding root $r(M_q)$.
3. Maintain the set of disks $\mathcal{D}$ that have already been placed in the cover in a data structure $\mathcal{S}$ that can efficiently test whether a query point is in the union of the disks in $\mathcal{D}$.
4. Search M in a bottom-up manner to find the next simplicial vertex q of G (whose root $r(M_q)$ is lowest in M, ignoring subtrees corresponding to points that are already covered). When the search reaches a root $r(M_q)$ we check whether q is in the union of the disks in $\mathcal{D}$. If so, we skip $r(M_q)$ and continue. Otherwise we add to $\mathcal{D}$ the maximal disk centered at $r(M_q)$, update $\mathcal{S}$ accordingly, and continue. At the end $\mathcal{D}$ contains the desired minimum disk cover.

We now give the details of the implementation of each of these steps. We first compute M in $O(n)$ time, using the algorithm of Chin et al. [10]. We regard the edges and vertices of $M \cup \partial P$ as the edges and vertices of a planar map H. We further partition H into "pseudo-trapezoids" (referred to as "trapezoids", in short), by connecting each vertex of M (lying in the interior of P, including breakpoints along edges which are equidistant from an edge of P and an endpoint of that edge) to its nearest points on ∂P.

For each point $q \in Q$ we compute the trapezoid $T(q)$ in the decomposition of H containing q. To do so we preprocess the decomposition of H in $O(n)$ time and construct the point location data structure of Kirkpatrick [24], which supports logarithmic-time point-location queries. Then we locate the trapezoids $T(q)$, for $q \in Q$, by m point location queries to this data structure, in $O(m \log n)$ time.

For each $q \in Q$, let e be the feature of P^* on $\partial T(q)$ (that is, $T(q)$ is contained in the Voronoi cell of e). We compute the closest point q' to q on e and take $A(q)$ to be the intersection of the line $q'q$ with M which is closest to q (and which also lies on $\partial T(q)$). It is easy to see that a maximal disk centered at $A(q)$ inside P contains q, and therefore M_q indeed contains $A(q)$.

We compute the roots $r(M_q)$, for $q \in Q$, as follows. We fix a root for M, for example, the rightmost vertex of P. We then traverse M in depth-first order from the root, and maintain its vertices on the stack (those vertices whose subtrees are still being traversed) in an array L, stored in their reverse order on the stack, i.e., in their order along the path of M from the root to the vertex currently being explored (L is in fact just a concrete structure for implementing the stack). When the search moves from a vertex v to a new vertex w, we insert w as the rightmost element of L, and when the search backtracks from a vertex v, we delete v from (the end of) L.

When we traverse the edge (v, w) during the depth-first search, we find $r(M_q)$ for every point q such that $A(q)$ is on the edge (v, w), as follows. First observe that testing whether a point $z \in M$ belongs to M_q is easy to do in $O(1)$ time, provided we know the feature (edge or vertex) of M containing z: This is equivalent to testing whether $|zq|$ is at most the radius of the maximal disk centered at z, which is readily available since we know the features of P^* nearest to z.

So let $q \in Q$ be a point whose anchor $A(q)$ lies on (v, w). If $v \notin M_q$ then $r(M_q)$ is on the edge (v, w). Furthermore, it is a point on (v, w) at equal distances to the two features of P defining the edge (v, w) and to q. We compute it by solving the appropriate system of algebraic equations (as is well known, there can be at most two solutions, for otherwise, since Voronoi regions are star-shaped, we would get an impossible planar embedding of $K_{3,3}$), and by taking the solution which lies on (v, w) closest to the root of M (i.e., to v).

If $v \in M_q$, we use binary search on the array L to find the farthest ancestor u of v which is still in M_q. The root $r(M_q)$ is on the edge from u to its parent, and we find it by solving a system of algebraic equations analogous to the one described above.

Having collected all the roots $r(M_q)$, we sort them along the edges of M containing them, and split the edges at these roots, making the roots additional vertices of M.

Finally we collect the roots which are centers of the maximal disks in our cover, using the greedy algorithm described above. For that we use a data structure $\mathcal{S}$ that maintains a set $\mathcal{D}$ of disks, subject to insertions of disks and queries of the form: Given a point q, determine whether q is in the union of the disks currently in $\mathcal{D}$. The structure $\mathcal{S}$ is initially empty, but, as we add disks to the cover $\mathcal{D}$, we insert them into $\mathcal{S}$.

We traverse M again bottom-up, stopping at each root $r(M_q)$. When we encounter a root $r(M_q)$, we use the data structure $\mathcal{S}$ to determine whether the corresponding point q lies in the union of the disks currently in $\mathcal{D}$. If the answer is yes, we continue traversing M, effectively ignoring q. Otherwise, we add to $\mathcal{D}$ the maximal disk centered at $r(M_q)$ and contained in P, update $\mathcal{S}$ accordingly, and continue. When the traversal terminates, $\mathcal{D}$ is the desired minimum disk cover.

We now analyze the complexity of the algorithm. Computing M and the Voronoi diagram H induced by $M \cup \partial P$, the triangulation of H into trapezoids, and the point location data structure for this triangulation, takes $O(n)$ time [10,24]. For each point $q \in Q$, computing $A(q)$ takes $O(\log n)$ time, for a total of $O(m \log n)$ time. The computation of the roots $r(M_q)$ takes $O(n + m \log n)$ time: maintaining the array L takes $O(n)$ time, and the binary searches for the roots take a total of $O(m \log n)$ time. Sorting the roots along the edges of M and subdividing M at these roots takes $O(n + m \log m)$ time.

Finally, in the last step we again traverse M and use the data structure $\mathcal{S}$ to determine, for each root $r(M_q)$, whether its corresponding point q is in the union of the disks that we have already added to the cover. We construct $\mathcal{S}$ using a standard reduction [3] that converts a static point location data structure to a dynamic incremental one, as follows. We recall that the combinatorial complexity of the of the boundary of the union of k disks is $O(k)$ [23], and that one can compute this union and preprocess it for logarithmic-time point location queries in time $O(k \log k)$. Let k be the number of disks currently in $\mathcal{D}$, and let $b_{\lfloor \log k \rfloor} \cdots b_1 b_0$ be the binary representation of k. For each i, $0 \le i \le \lfloor \log k \rfloor$, such that $b_i = 1$, $\mathcal{D}$ contains a static point location data structure for the union of a subset of 2^i disks. Moreover, each of the k disks belongs to exactly one of these subsets. Thus, given a query point q, one can determine in $O(\log^2 k)$ time whether q lies in the union of the k disks, by performing a point location query in at most $\lfloor \log k \rfloor + 1$ substructures. To insert a new disk into $\mathcal{S}$, we construct a new static data structure containing the new disk and all the disks in the subsets corresponding to the maximal block of least significant bits $b_0, b_1, \ldots$, which are equal to 1 in the binary representation of the current k. This is analogous to performing an increment of the binary counter $b_{\lfloor \log k \rfloor} \cdots b_1 b_0$. Since each disk participates in the construction of at most $O(\log k)$ static structures, an insertion takes $O(\log^2 k)$ amortized time. In summary, we obtain:

Theorem 2. *Let P be a simple polygon with n edges and Q a set of m points contained in P. We can compute a minimum cover of Q by disks contained in P in $O(n + m(\log n + \log^2 m))$ time and $O(n + m)$ space.*

Remark. If the minimum cover consists of $k \ll m$ disks, the running time improves to $O(n + m(\log n + \log m + \log^2 k))$. Hence, if $k = O(1)$, say, or if we just want to decide whether Q can be covered by at most $k = O(1)$ disks, the superlinearity of the bound is caused only by the steps which compute the anchor points $A(q)$ and the roots $r(M_q)$, for $q \in Q$. It is a challenging open problem to come up with an alternative approach which avoids the supelinear cost of these steps.

3 Covering by Homothetic Copies of a Convex Set

Let P and Q be as in Section 2, and let $\mathcal{O}$ be some fixed compact convex set with nonempty interior. For a large part of the analysis, this is essentially all we assume about $\mathcal{O}$. For the algorithmic part, however, we need to assume that $\mathcal{O}$ has a sufficiently simple shape so as to facilitate efficient implementation of certain operations on $\mathcal{O}$ as well as efficient construction of a dynamic point location data structure, similar to the structure $\mathcal{S}$ used above. For the time being, we only add the assumption that P, Q and $\mathcal{O}$ are in general position, to avoid possible degeneracies in the constructs that extend those studied in the preceding section. Further assumptions will be elaborated below.

We fix some point o inside $\mathcal{O}$ as its "center point", and assume that $\mathcal{O}$ is initially specified so that o lies at the origin. Thinking of each point of $\mathcal{O}$ as a vector, $\lambda\mathcal{O}$ then denotes the convex set obtained from $\mathcal{O}$ by scaling it about o by λ, for any positive λ. The *convex distance function* induced by $\mathcal{O}$ is $d_{\mathcal{O}}(p, q) = \inf\{\lambda \mid q \in p + \lambda\mathcal{O}\}$. We refer to it as the *$\mathcal{O}$-distance.* Chew and Drysdale [8] were the first to study Voronoi diagrams under convex distance functions; see also Leven and Sharir [25]. Note that $d_{\mathcal{O}}$ is a metric if and only if $\mathcal{O}$ is centrally symmetric with respect to its center.

The medial axis $M_{\mathcal{O}}$ of P under the $\mathcal{O}$-distance is defined analogously to the medial axis of P under the Euclidean distance, as the locus of all points inside P whose $\mathcal{O}$-distance to the boundary is attained in at least two points. Assuming general position, $M_{\mathcal{O}}$ is a connected 1-dimensional network, consisting of vertices and edges. Each edge is the locus of all points which are at the same (nearest) $\mathcal{O}$-distance from two features of P^* (where P^* is defined as in the previous section). A vertex of $M_{\mathcal{O}}$ which is not a vertex of P is a point at the same (nearest) $\mathcal{O}$-distance from three features of P^*. The shape of the edges of $M_{\mathcal{O}}$ depends on the shape of $\mathcal{O}$. For example, if $\mathcal{O}$ is a convex polygon then each edge is a polygonal curve, whose breakpoints correspond to placements of the center o of $\mathcal{O}$ at which a vertex of $\mathcal{O}$ touches a vertex of P. See [8,25] for more details. Finally, as in the previous section, it follows from basic properties of generalized Voronoi diagrams that $M_{\mathcal{O}}$ is a tree.

As in the preceding section, for a point $q \in Q$, we denote by M_q the portion of $M_{\mathcal{O}}$ consisting of centers of maximal homothets of $\mathcal{O}$ that contain q (and are contained in P). Lemma 3 below is analogous to Lemma 1, asserting that for each $q \in Q$, M_q is a subtree of $M_{\mathcal{O}}$; its proof can be found in the full version of this paper.[3] The analysis below uses the well known fact that homothetic copies of Q (in general position) are *pseudo-disks*; see, e.g., [23].

Lemma 3. *M_q is connected for each point $q \in Q$.*

Consider the graph $G_{\mathcal{O}}$ defined similarly to G in the previous section. Its vertices are the points of Q, and it contains an edge between two points $p, q \in Q$ if there exists a homothet of $\mathcal{O}$ containing both p and q and contained in P. As in the case of disks, one can show that there is such a homothet if and only if there is a homothet containing p and q whose center is on $M_{\mathcal{O}}$ and whose boundary touches ∂P (at least twice).

Again, if we identify each point q of Q with its subtree M_q, then $G_{\mathcal{O}}$ is the intersection graph of these subtrees and thus $G_{\mathcal{O}}$ is a chordal graph. As in Section 2, since pairwise intersecting subtrees of a tree T have a point in common, covering Q by homothets of $\mathcal{O}$ is equivalent to a clique cover of $G_{\mathcal{O}}$, so our problem now is to find a minimum clique cover of $G_{\mathcal{O}}$.

We use the same high-level algorithm as in Section 2. Below, we mainly refer to the steps of the algorithm that are affected by the use of $\mathcal{O}$-distance instead of Euclidean distance. Recall that so far the analysis did not require any further assumptions concerning the actual shape of $\mathcal{O}$. However, to facilitate an efficient implementation of the algorithm, we need to assume that this shape is sufficiently simple, so as to allow various operations on $\mathcal{O}$ and on a constant number of other features (points and/or line segments) to be performed in constant time. Examples of such operations are computing the $\mathcal{O}$-distance between two points, or between a point and a line segment, finding a point at the same $\mathcal{O}$-distance from three features of P^*, etc. The simplest way to enforce these properties is to assume that $\mathcal{O}$ has *constant description complexity* (see, e.g., [31]).

The combinatorial complexity of the medial axis $M_{\mathcal{O}}$ is $O(n)$, we compute it in time $O(n)$ using the algorithm of Chin et al. [10] (which also applies to convex distance functions). The planar map H (induced by $M_{\mathcal{O}} \cup \partial P$) is partitioned, in time $O(n)$, into simply-shaped cells, by connecting each vertex of $M_{\mathcal{O}} \setminus \partial P$ to its nearest point(s) (in the $\mathcal{O}$-distance) on ∂P.

As in the previous section, we perform, in total time $O(m \log n)$, point location queries in H for the points in Q, as in [24]. Let $T(q)$ be the trapezoid containing a point $q \in Q$, and let e be the feature of P^* on $\partial T(q)$. We compute the closest point q' to q on e under the $\mathcal{O}$-distance, and take $A(q)$ to be the intersection of the line $q'q$ with $M_{\mathcal{O}}$ which is closest to q (and which also lies on $\partial T(q)$). It is easy to see, arguing as above, that the maximal copy of $\mathcal{O}$ centered at $A(q)$ and contained in P touches ∂P at q' (and at another point) and contains q, and therefore M_q contains $A(q)$. Computing the roots $r(M_q)$ and sorting them along the edges of $M_{\mathcal{O}}$, is done exactly as in the previous section, in $O(n + m(\log n + \log m))$ time.

Finally, in the last step, we maintain, in a data structure $\mathcal{S}$, the union of all the homothets of $\mathcal{O}$ that have so far been placed in the cover, and perform on it point location queries. Since homothets of $\mathcal{O}$ are pseudo-disks, we can use essentially the same structure described in the previous section, with the same time and space bounds. We thus obtain (the remark following Theorem 2 applies here as well):

Theorem 4. *Let $\mathcal{O}$ be a fixed compact convex set of constant description complexity, let P be a simple polygon with n edges, and let Q be a set of m points in P. We can compute a minimum cover of Q by homothets of $\mathcal{O}$ contained in P, in $O(n + m(\log n + \log^2 m))$ time, in an appropriate model of computation, using $O(n + m)$ storage.*

4 Covering by Disks in a Sufficiently Narrow Annulus

Assume that the points of Q lie in an annulus R rather than in a simple polygon. In this case, M, the medial axis of R, is not a tree but a circle. Specifically, let c

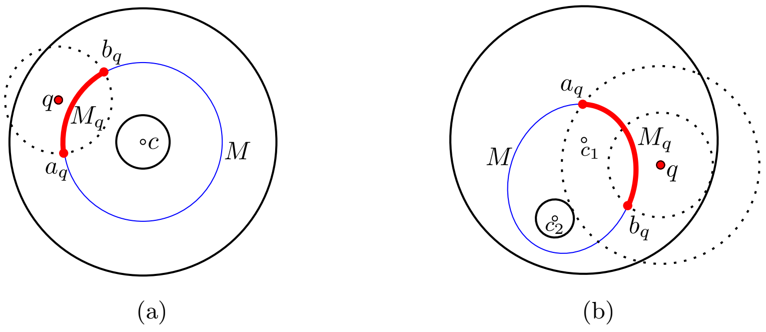


Fig. 2. (a) The arc M_q of a point $q \in Q$, obtained as the intersection of M with the (dotted) disk of radius $\frac{r_2-r_1}{2}$ around q. (b) R is a disk centered at c_1 with a hole centered at c_2; the medial axis M is elliptic. The endpoints of M_q are at equal distance to the two circles bounding R and its hole and to q.

be the center of R and let r_1 and r_2 be inner and outer radii of R, respectively, then M is a circle of radius $r_M = \frac{r_1+r_2}{2}$, centered at c.

As in the previous sections, each point $q \in Q$ is associated with an arc M_q of the circle M, which is the portion of M consisting of centers of maximal disks that cover q (and are contained in R). See Fig. 2(a), which illustrates a proof of the property that M_q is the intersection of M with the disk of radius $\frac{r_2-r_1}{2}$ centered at q. Again, we consider the intersection graph G of these arcs. However, unlike the previous cases, G is not chordal, since M is not a tree. Instead, G is a circular-arc graph (see, e.g., [13]). Moreover, we argue that if $r_2 < \frac{2+\sqrt{3}}{2-\sqrt{3}}r_1 \approx 13.93r_1$ then G has the 2-Helly property, which means that the circular arcs in a clique of G have a point in common.

Indeed, since M_q is the intersection of M with the disk of radius $\frac{r_2-r_1}{2}$ centered at q, it is maximal when a_q, q, and b_q are collinear, where a_q and b_q are the endpoints of M_q. Let $\theta = \angle a_q c b_q$. If a_q, q, and b_q are indeed collinear then $\sin\frac{\theta}{2} = \frac{r_2-r_1}{r_2+r_1}$. So if $r_2 < \frac{2+\sqrt{3}}{2-\sqrt{3}}r_1$ then $\sin\frac{\theta}{2} < \frac{\sqrt{3}}{2}$, and therefore $\theta < 2\pi/3$. Consider a clique $\mathcal{C}$ in G. Since all arcs are of length smaller than 1/3 the length of M, and they all intersect one specific arc of $\mathcal{C}$, then they cannot cover M completely. Consequently, $\mathcal{C}$ can be viewed as a set of pairwise intersecting intervals on a line and it follows that all the arcs of $\mathcal{C}$ must have a common intersection.

We conclude that, again, a cover of Q by disks contained in R corresponds to a clique cover of G, and vice versa. We thus compute a minimum clique cover of G by applying the $O(m)$-time algorithm of Hsu and Tsai [18], after sorting the arcs M_q by their endpoints in $O(m \log m)$ time. In summary, we obtain

Theorem 5. *Let R be an annulus such that $r_2 < \frac{2+\sqrt{3}}{2-\sqrt{3}}r_1$, where r_1, r_2 are the inner and outer radii of R, respectively, and let Q a set of m points contained in R. We can compute a minimum cover of Q by disks contained in R in $O(m \log m)$ time and $O(m)$ space.*

Theorem 5 also applies to the slightly more general case, where R is a disk with a circular hole, not necessarily concentric; see Fig. 2(b). In this case, the medial axis is an ellipse with foci at the centers of R and of its hole. For each $q \in Q$, M_q is still a connected arc of M. Specifically, the endpoints of M_q are the points at equal distance to the two circles bounding R and its hole and to q, and there can be at most two such points. (Otherwise, arguing as in Section 2, we would get an impossible planar embedding of $K_{3,3}$.)

The graph G is a circular-arc graph, and it possesses the 2-Helly property provided that the hole is not too small. (The exact condition is that there do not exist three points $q_1, q_2, q_3 \in R$ whose arcs M_{q_1}, M_{q_2}, M_{q_3} cover M.) Hence, if this condition holds then a minimum clique cover can be found as above, with the same asymptotic bounds on the running time and storage.

Finally, we do not know how critical is the assumption that R is not too wide, as in Theorem 5. Does the problem become hard if R is wider?

Acknowledgments. The authors wish to thank Lior Kapelushnik and an anonymous referee for independently suggesting that we use the data structure for union of disks instead of a less efficient range reporting data structure that was used in a previous version of this paper.

References

1. Agarwal, P.K., Efrat, A., Sharir, M.: Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. SIAM J. Comput. 29, 912–953 (2000)
2. Brönnimann, H., Goodrich, M.T.: Almost optimal set covers in finite vc-dimension. Discrete Comput. Geom. 14(4), 469–479 (1995)
3. Bentley, J.L., Saxe, J.B.: Decomposable searching problems I: Static-to-dynamic transformation. J. Algorithms 1(4), 301–358 (1980)
4. Buneman, P.: A characterization of rigid circuit graphs. Discrete Math. 9, 205–212 (1974)
5. Calinescu, G., Mandoiu, I.I., Wan, P.-J., Zelikovsky, A.: Selecting forwarding neighbors in wireless ad hoc networks. MONET 9(2), 101–111 (2004)
6. Carmi, P., Katz, M.J., Lev-Tov, N.: Covering points by unit disks of fixed location. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 644–655. Springer, Heidelberg (2007)
7. Chan, T.M.: A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. J. ACM 57(3), Article 16 (2010)
8. Chew, L.P., Drysdale III., R.L.: Voronoi diagrams based on convex distance functions. In: SCG 1985: Proc. First Annual Sympos. Comput. Geom., pp. 235–244 (1985)
9. Chiang, Y., Tamassia, R.: Dynamic algorithms in computational geometry. Proc. IEEE 80(9), 1412–1434 (1992)
10. Chin, F.Y.L., Snoeyink, J., Wang, C.A.: Finding the medial axis of a simple polygon in linear time. Discrete Comput. Geom. 21(3), 405–420 (1999)
11. Gavril, F.: Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. SIAM J. Comput. 1(2), 180–187 (1972)

12. Gavril, F.: The intersection graphs of subtrees of a tree are exactly the chordal graphs. J. Combinat. Theory Ser. B 16, 47–56 (1974)
13. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)
14. Grötschel, M., Lovász, L., Schrijver, A.: Polynomial algorithms for perfect graphs. In: Berge, C., Chvátal, V. (eds.) Topics on Perfect Graphs. Ann. Discrete Math., vol. 21, pp. 325–356. North-Holland, Amsterdam (1984)
15. Halperin, D., Linhart, C.: The minimum enclosing disk with obstacles (1999) (Manuscript)
16. Halperin, D., Sharir, M., Goldberg, K.Y.: The 2-center problem with obstacles. J. Algorithms 42(1), 109–134 (2002)
17. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. J. ACM 32(1), 130–136 (1985)
18. Hsu, W.L., Tsai, K.H.: Linear time algorithms on circular-arc graphs. Inform. Process. Lett. 40(3), 123–129 (1991)
19. Hurtado, F., Sacristán, V., Toussaint, G.: Some constrained minimax and maximin location problems. Studies in Locational Analysis 15, 17–35 (2000)
20. Katz, M.J., Morgenstern, G.: A scheme for computing minimum covers within simple regions. In: Dehne, F., et al. (eds.) 11th Int. Sympos. Algorithms and Data Structures (WADS). LNCS, vol. 5664, pp. 447–458. Springer, Heidelberg (2009)
21. Katz, M.J., Morgenstern, G.: Guarding orthogonal art galleries with sliding cameras. In: 25th European Workshop on Comput. Geom, pp. 159–162 (2009)
22. Katz, M.J., Roisman, G.S.: On guarding the vertices of rectilinear domains. Comput. Geom. 39(3), 219–228 (2008)
23. Kedem, K., Livne, R., Pach, J., Sharir, M.: On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. Discrete Comput. Geom. 1(1), 59–71 (1986)
24. Kirkpatrick, D.: Optimal search in planar subdivisions. SIAM J. Comput. 12(1), 28–34 (1983)
25. Leven, D., Sharir, M.: Planning a purely translational motion for a convex object in two–dimensional space using generalized Voronoi diagrams. Discrete Comput. Geom. 2, 9–31 (1987)
26. Mehlhorn, K., Näher, S.: Dynamic fractional cascading. Algorithmica 5(2), 215–241 (1990)
27. Motwani, R., Raghunathan, A., Saran, H.: Perfect graphs and orthogonally convex covers. SIAM J. Discrete Math. 2(3), 371–392 (1989)
28. Motwani, R., Raghunathan, A., Saran, H.: Covering orthogonal polygons with star polygons: The perfect graph approach. J. Comput. Syst. Sci. 40(1), 19–48 (1990)
29. Mustafa, N.H., Ray, S.: PTAS for geometric hitting set problems via local search. In: SCG 2009: Proc. 25th Annual Sympos. Comput. Geom, pp. 17–22 (2009)
30. Narayanappa, S., Vojtechovský, P.: An improved approximation factor for the unit disk covering problem. In: 18th Canadian Conf. on Comput. Geom, pp. 15–18 (2006)
31. Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and their Geometric Applications. Cambridge University Press, New York (1995)
32. Worman, C., Keil, J.M.: Polygon decomposition and the orthogonal art gallery problem. Int. J. Comput. Geometry Appl. 17(2), 105–138 (2007)

Stability of ε-Kernels

Pankaj K. Agarwal, Jeff M. Phillips, and Hai Yu

Duke University, University of Utah, and Google

Abstract. Given a set P of n points in $\mathbb{R}^d$, an ε-kernel $K \subseteq P$ approximates the directional width of P in every direction within a relative $(1-\varepsilon)$ factor. In this paper we study the stability of ε-kernels under dynamic insertion and deletion of points to P and by changing the approximation factor ε. In the first case, we say an algorithm for dynamically maintaining a ε-kernel is stable if at most $O(1)$ points change in K as one point is inserted or deleted from P. We describe an algorithm to maintain an ε-kernel of size $O(1/\varepsilon^{(d-1)/2})$ in $O(1/\varepsilon^{(d-1)/2} + \log n)$ time per update. Not only does our algorithm maintain a stable ε-kernel, its update time is faster than any known algorithm that maintains an ε-kernel of size $O(1/\varepsilon^{(d-1)/2})$. Next, we show that if there is an ε-kernel of P of size κ, which may be dramatically less than $O(1/\varepsilon^{(d-1)/2})$, then there is an $(\varepsilon/2)$-kernel of P of size $O(\min\{1/\varepsilon^{(d-1)/2}, \kappa^{\lfloor d/2 \rfloor} \log^{d-2}(1/\varepsilon)\})$. Moreover, there exists a point set P in $\mathbb{R}^d$ and a parameter $\varepsilon > 0$ such that if every ε-kernel of P has size at least κ, then any $(\varepsilon/2)$-kernel of P has size $\Omega(\kappa^{\lfloor d/2 \rfloor})$.[1]

1 Introduction

With recent advances in sensing technology, massive geospatial data sets are being acquired at an unprecedented rate in many application areas, including GIS, sensor networks, robotics, and spatial databases. Realizing the full potential of these data sets requires developing scalable algorithms for analyzing and querying them. Among many interesting algorithmic developments to meet this challenge, there is an extensive amount of work on computing a "small summary" of large data sets that preserves certain desired properties of the input data and on obtaining a good trade-off between the quality of the summary and its size. A coreset is one example of such approximate summaries. Specifically, for an input set P and a function f, a *coreset* $C \subseteq P$ is a subset of P (with respect to f) with the property that $f(C)$ approximates $f(P)$. If a small-size coreset C can be computed quickly (much faster than computing $f(P)$), then one can compute an approximate value of $f(P)$ by first computing C and then computing $f(C)$. This coreset-based approach has been successfully used in a wide range of geometric optimization problems over the last decade; see [2].

[1] Research supported by subaward CIF-32 from NSF grant 0937060 to CRA, by NSF under grants CNS-05-40347, CFF-06-35000, and DEB-04-25465, by ARO grants W911NF-04-1-0278 and W911NF-07-1-0376, by an NIH grant 1P50-GM-08183-01, by a DOE grant OEG-P200A070505, and by a grant from the U.S.–Israel Binational Science Foundation.

ε-kernels. Agarwal *et al.* [1] introduced the notion of ε-kernels and proved that it is a coreset for many functions. For any direction $u \in \mathbb{S}^{d-1}$, let $P[u] = \arg\max_{p \in P} \langle p, u \rangle$ be the extreme point in P along u; $\omega(P, u) = \langle P[u] - P[-u], u \rangle$ is called the *directional width* of P in direction u. For a given $\varepsilon > 0$, $K \subset P \subset \mathbb{R}^d$ is called an *ε-kernel* of P if

$$\langle P[u] - K[u], u \rangle \leq \varepsilon \omega(P, u)$$

for all directions $u \in \mathbb{S}^{d-1}$.[2] For simplicity, we assume $\varepsilon \in (0, 1)$, because for $\varepsilon \geq 1$, one can choose a constant number of points to form an ε-kernel. By definition, if X is an ε-kernel of P and K is a δ-kernel of X, then K is a $(\delta + \varepsilon)$-kernel of P.

Agarwal et al. [1] showed that there exists an ε-kernel of size $O(1/\varepsilon^{(d-1)/2})$ and it can be computed in time $O(n + 1/\varepsilon^{3d/2})$, when d is fixed (assumed throughout the paper). The running time was improved by Chan [6] to $O(n + 1/\varepsilon^{d-3/2})$ (see also [10]). In a number of applications, the input point set is being updated periodically, so algorithms have also been developed to maintain ε-kernels dynamically. Agarwal *et al.* [1] had described a data structure to maintain an ε-kernel of size $O(1/\varepsilon^{(d-1)/2})$ in $(\log(n)/\varepsilon)^{O(d)}$ time per update. The update time was recently improved by Chan [7] to $O((1/\varepsilon^{(d-1)/2}) \log n + 1/\varepsilon^{d-3/2})$. His approach can also maintain an ε-kernel of size $O((1/\varepsilon^d) \log n)$ with update time $O(\log n)$. If only insertions are allowed (e.g. in a streaming model), the size of the data structure can be improved to $O(1/\varepsilon^{(d-1)/2})$ [4,11].

In this paper we study two problems related to the *stability* of ε-kernels: how ε-kernels change as we update the input set or vary the value of ε.

Dynamic stability. Since the aforementioned dynamic algorithms for maintaining an ε-kernel focus on minimizing the size of the kernel, changing a single point in the input set P may drastically change the resulting kernel. This is particularly undesirable when the resulting kernel is used to build a dynamic data structure for maintaining another information. For example, kinetic data structures (KDS) based on coresets have been proposed to maintain various extent measures of a set of moving points [2]. If an insertion or deletion of an object changes the entire summary, then one has to reconstruct the entire KDS instead of locally updating it. In fact, many other dynamic data structures for maintaining geometric summaries also suffer from this undesirable property [9].

We call an ε-kernel *s-stable* if the insertion or deletion of a point causes the ε-kernel to change by at most s points. For brevity, if $s = O(1)$, we call the ε-kernel to be *stable*. Chan's dynamic algorithm can be adapted to maintain a stable ε-kernel of size $O((1/\varepsilon^{d-1}) \log n)$; see Lemma 1 below. An interesting question is whether there is an efficient algorithm for maintaining a stable ε-kernel of size $O(1/\varepsilon^{(d-1)/2})$, as points are being inserted or deleted. Maintaining a stable ε-kernel dynamically is difficult for two main reasons. First, for an input set P,

[2] This is a slightly stronger version of the definition than defined in [1] and an ε-kernel K gives a relative $(1 + 2\varepsilon)$-approximation of $\omega(P, u)$ for all $u \in \mathbb{S}^{d-1}$ (i.e. $\omega(K, u) \leq \omega(P, u) \leq (1 + 2\varepsilon)\omega(K, u)$).

many algorithms compute ε-kernels in two or more steps. They first construct a large ε-kernel K' (e.g. see [1,7]), and then use a more expensive algorithm to create a small ε-kernel of K'. However, if the first algorithm is unstable, then K' may change completely each time P is updated. Second, all of the known ε-kernel algorithms rely on first finding a "rough shape" of the input set P (e.g., finding a small box that contains P), estimating its fatness [5]. This rough approximation is used crucially in the computation of the ε-kernel. However, this shape is itself very unstable under insertions or deletions to P. Overcoming these difficulties, we prove the following in Section 2:

Theorem 1. *Given a parameter $0 \le \varepsilon \le 1$, a stable ε-kernel of size $O(1/\varepsilon^{(d-1)/2})$ of a set of n points in $\mathbb{R}^d$ can be maintained under insertions and deletions in $O(1/\varepsilon^{(d-1)/2} + \log n)$ amortized time.*

Note that the update time of maintaining an ε-kernel of size $O(1/\varepsilon^{(d-1)/2})$ is better than that in [7].

Approximation stability. If the size of an ε-kernel K is $O(1/\varepsilon^{(d-1)/2})$, then decreasing ε changes K quite predictably. However, this is the worst-case bound, and it is possible that the size of K may be quite small, e.g., $O(1)$, or in general much smaller than the $1/\varepsilon^{(d-1)/2}$ maximum (efficient algorithms are known for computing ε-kernels of near-optimal size [2]). Then how much can the size increase as we reduce the allowable error from ε to $\varepsilon/2$? For any $\varepsilon > 0$, let $\kappa(P, \varepsilon)$ denote the minimum size of an ε-kernel of P. Unlike many shape simplification problems, in which the size of simplification can change drastically as we reduce the value of ε, we show (Section 3) that this does not happen for ε-kernels and that $\kappa(P, \varepsilon/2)$ can be expressed in terms of $\kappa(P, \varepsilon)$.

Theorem 2. *For any point set P and for any $\varepsilon > 0$,*

$$\kappa(P, \varepsilon/2) = O(\min\{\kappa(P, \varepsilon)^{\lfloor d/2 \rfloor} \log^{d-2}(1/\varepsilon), 1/\varepsilon^{(d-1)/2}\}).$$

Moreover, there exist a point set P and some $\varepsilon > 0$ such that $\kappa(P, \varepsilon/2) = \Omega(\kappa(P, \varepsilon)^{\lfloor d/2 \rfloor})$.

2 Dynamic Stability

In this section we describe an algorithm that proves Theorem 1. The algorithm is composed of a sequence of modules, each with certain property. We first state that Chan's dynamic coreset algorithm [7] can be made stable (see proof in full version [3]):

Lemma 1. *For any $0 < \varepsilon < 1$, an ε-kernel K of P of size $O((1/\varepsilon^{d-1}) \log n)$ can be maintained in $O(\log n)$ time with $O(1)$ changes to K per update.*

We first define the notion of anchor points and fatness of a point set and describe two algorithms for maintaining stable ε-kernels with respect to a fixed anchor: one of them maintains a kernel of size $O(1/\varepsilon^{d-1})$ and the other of size $O(1/\varepsilon^{(d-1)/2})$; the former has smaller update time. Then we describe the algorithm for updating anchor points and maintaining a stable kernel as the anchors change. Finally, we put these modules together to obtain the final algorithm. We make the following simple observation, which will be crucial for combining different modules.

Lemma 2 (Composition Lemma). *If K is an s-stable ε-kernel of P and K' is an s'-stable ε'-kernel of K, then K' is an $(s \cdot s')$-stable $(\varepsilon + \varepsilon')$-kernel of P.*

Anchors and fatness of a point set. We call a point set P *β-fat* if

$$\max_{u,v \in \mathbb{S}^{d-1}} \omega(P,u)/\omega(P,v) \leq \beta.$$

If β is a constant, we sometimes just say that P is *fat*. An arbitrary point set P can be made fat by applying an affine transform: we first choose a set of $d+1$ *anchor points* $A = \{a_0, a_1, \ldots, a_d\}$ using the following procedure of Barequet and Har-Peled [5]. Choose a_0 arbitrarily. Let a_1 be the farthest point from a_0. Then inductively, let a_i be the farthest point from the flat $\mathrm{span}(a_0, \ldots, a_{i-1})$. (See Figure 1.) The anchor points A define a bounding box I_A with center at a_0 and orthogonal directions defined by vectors from the flat $\mathrm{span}(a_0, \ldots, a_{i-1})$ to a_i. The extents of I_A in each orthogonal direction is defined by placing each a_i on a bounding face and extending I_A the same distance from a_0 in the opposite direction. Next we perform an affine transform T_A on P such that the vector from the flat $\mathrm{span}(a_0, \ldots, a_{i-1})$ to a_i is equal to e_i, where $e_0 = (0, \ldots, 0), e_1 = (1, 0, \ldots, 0), \ldots, e_d = (0, \ldots, 0, 1)$. This ensures that $T_A(P) \subseteq T_A(I_A) = [-1,1]^d$. The next lemma shows that $T_A(P)$ is fat, and follows easily from [8].

Lemma 3. *For all $u \in \mathbb{S}^{d-1}$ and for $\beta_d \leq 2^d d^{5/2} d!$,*

$$\omega(T_A(A),u) \leq \omega(T_A(P),u) \leq \omega(T_A(I_A),u) \leq \beta_d \cdot \omega(T_A(A),u). \qquad (1)$$

Agarwal *et al.* [1] show if K is an ε-kernel of P, then $T(K)$ is an ε-kernel of $T(P)$ for any affine transform T, which implies that one can compute an ε-kernel of $T(P)$. We will need the following generalization of the definition of ε-kernel. For two points sets P and Q, a subset $K \subseteq P$ is called an *ε-kernel of P with respect to Q* if $\langle P[u] - K[u], u\rangle \leq \varepsilon\omega(Q,u)$ for all $u \in \mathbb{S}^{d-1}$.

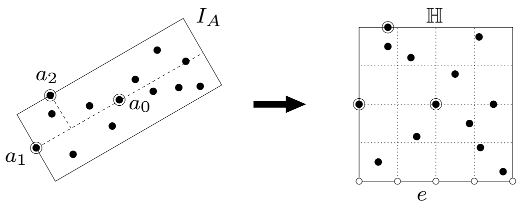


Fig. 1. Anchor points $A = \{a_0, a_1, a_2\}$, rectangle I_A, and transform T_A applied to P; square $\mathbb{H}$, two-dimensional grid $\mathbb{G}$, and one-dimensional grid $\mathbb{G}_e$ on the edge e of $\mathbb{H}$

Stable ε-kernels for a fixed anchor. Let A be a set of anchor points of P, as described above. We describe algorithms for maintaining stable ε-kernels (with respect to A) under the assumption that A remains a set of anchor points of P, i.e., $A \subseteq P \subset I_A$, as P is being updated by inserting and deleting points. In view of the above discussion, without loss of generality, we assume $I_A = [-1, +1]^d$ and denote it by $\mathbb{H}$. As for the static case [1,6], we first describe a simpler algorithm that maintains a stable ε-kernel of size $O(1/\varepsilon^{d-1})$, and then a more involved one that maintains a stable ε-kernel of size $O(1/\varepsilon^{(d-1)/2})$.

Set $\delta = \varepsilon/\sqrt{d}$ and draw a d-dimensional grid $\mathbb{G}$ inside $\mathbb{H}$ of size δ, i.e., the side-length of each grid cell is at most δ; $\mathbb{G}$ has $O(1/\delta^d)$ cells. For each grid cell τ, let $P_\tau = P \cap \tau$. For a point $x \in \mathbb{H}$ lying in a grid cell τ, let $\hat{x}$ be the vertex of τ nearest to the origin; we can view x being *snapped* to the vertex $\hat{x}$. For each facet f of $\mathbb{H}$, $\mathbb{G}$ induces a $(d-1)$-dimensional grid $\mathbb{G}_f$ on f; $\mathbb{G}$ contains a *column* of cells for each cell in $\mathbb{G}_f$. For each cell $\Delta \in \mathbb{G}_f$, we choose (at most) one point of P as follows: let τ be the nonempty grid cell in the column of $\mathbb{G}$ corresponding to Δ that is closest to f. We choose an arbitrary point from P_τ; if there is no nonempty cell in the column, no point is chosen. Let L_f be the set of chosen points. Set $\mathcal{L} = \bigcup_{f \in \mathbb{H}} L_f$. Agarwal *et al.* [1] proved that $\mathcal{L}$ is an ε-kernel of P. Insertion or deletion of a point in P affects at most one point in L_f, and it can be updated in $O(\log(1/\varepsilon))$ time. Hence, we obtain the following:

Lemma 4. *Let P be a set of n points in $\mathbb{R}^d$, let $A \subseteq P$ be a set of anchor points of P, and let $0 < \varepsilon < 1$ be a parameter. P can be preprocessed in $O(n + 1/\varepsilon^{d-1})$ time, so that a (2d)-stable ε-kernel of P with respect to A of size $O(1/\varepsilon^{d-1})$ can be maintained in $O(\log 1/\varepsilon)$ time per update provided that A remains an anchor set of P.*

Agarwal *et al.* [1] and Chan [6] have described algorithms for computing an ε-kernel of size $O(1/\varepsilon^{(d-1)/2})$. We adapt Chan's algorithm to maintain a stable ε-kernel with respect to a fixed anchor A. We begin by mentioning a result of Chan that lies at the heart of his algorithm.

Lemma 5 (Chan [6]). *Let $E \in \mathbb{N}$, $E^\tau \leq F \leq E$ for some $0 < \tau < 1$, and $P \subseteq [0 : E]^{d-1} \times \mathbb{R}$ a set of at most n points. For all grid points $b \in [0 : F]^{d-1} \times \mathbb{R}$, the nearest neighbors of each b in P can be computed in time $O(n + E^{d-2}F)$.*

We now set $\gamma = \sqrt{\varepsilon}/c$ for a constant $c > 1$ to be used in a much sparser grid than with δ. Let $\mathbb{C} = [-2, +2]^d$ and f be a facet of $\mathbb{C}$. We draw a $(d-1)$-dimensional grid on f of size γ. Assuming f lies on the plane $x_d = -2$, we choose a set $B_f = \{(i_1\gamma, \ldots, i_{d-1}\gamma, -2) \in \mathbb{Z}^d \mid -\lceil 2/\gamma \rceil \leq i_1, \ldots, i_{d-1} \leq \lceil 2/\gamma \rceil\}$ of grid points. For a subset $X \subseteq P$ and a point b, we define $\psi(X, b) = \arg\min_{x \in X} \|\hat{x} - b\|$, i.e., the point in X such that the snapped point is nearest to b. For a set R, $\psi(X, R) = \{\psi(X, r) \mid r \in R\}$. There is a one to one mapping between the faces of $\mathbb{C}$ and $\mathbb{H}$, so we also use f to denote the corresponding facet of $\mathbb{H}$. Let L_f be the set of points chosen in the previous algorithm corresponding to facet f of $\mathbb{H}$ for computing an $(\varepsilon/2)$-kernel of P. Set $G_f = \psi(L_f, B_f)$. Chan showed that $\mathcal{G} = \bigcup_{f \in \mathbb{C}} G_f$ is an $(\varepsilon/2)$-kernel of $\mathcal{L}$ and thus an ε-kernel of P. Scaling $\mathbb{G}$ and

B_f appropriately and using Lemma 5, G_f can be computed in $O(n+1/\varepsilon^{d-3/2})$ time. Hence, $\mathcal{G}$ can be computed in $O(n+1/\varepsilon^{d-3/2})$ time.

Note that $\psi(L_f, b)$ can be the same for many points $b \in B_f$, so insertion or deletion of a point in P (and thus in L_f) may change G_f significantly, thereby making $\mathcal{G}$ unstable. We circumvent this problem by introducing two new ideas. First, $\psi(L_f, B_f)$ is computed in two stages, and second it is computed in an iterative manner. We describe the construction and the update algorithm for f; the same algorithm is repeated for all facets.

We partition $\mathbb{H}$ into $O(1/\gamma^{d-1})$ boxes: for $J = \langle i_1 \ldots, i_{d-1} \rangle \in [-1/\gamma, 1/\gamma]^{d-1} \cap \mathbb{Z}^{d-1}$, we define $\mathbb{H}_J = [i_1\gamma, (i_1+1)\gamma] \times \cdots \times [i_{d-1}\gamma, (i_{d-1}+1)\gamma] \times [-1, +1]$. We maintain a subset $X \subseteq L_f$. Initially, we set $X = L_f$. Set $X_J = X \cap \mathbb{H}_J$. We define a total order on the points of B_f. Initially, we sort B_f in lexicographic order, but the ordering will change as insertions and deletions are performed on P. Let $\langle b_1, \ldots, b_u \rangle$ be the current ordering of B_f. We define a map $\varphi : B_f \to L_f$ as follows. Suppose $\varphi(b_1), \ldots, \varphi(b_{i-1})$ have been defined. Let $J_i = \arg\min_J \|\hat{\psi}(X_J, b_i) - b_i\|$; here $\hat{\psi}(\cdot)$ denotes the snapped point of $\psi(\cdot)$. We set $\varphi(b_i) = \psi(X_{J_i}, b_i)$. We delete $\varphi(b_i)$ from X (and from X_{J_i}) and recompute $\hat{\psi}(X_{J_i}, B_f)$. Set $K_f = \{\varphi(b) \mid b \in B_f\}$ and $K = \bigcup_f K_f$. Computing J_i and $\varphi(b_i)$ takes $O(1/\varepsilon^{(d-1)/2})$ time, and, by Lemma 5, $\psi(X_{J_i}, B_f)$ can be computed in $O(|X_J| + 1/\gamma^{d-2} \cdot 1/\gamma) = O(1/\varepsilon^{(d-1)/2})$ time.

It can be proved that the map φ and the set K_f satisfy the following properties:

(P1) $\varphi(b_i) \neq \varphi(b_j)$ for $i \neq j$,
(P2) $\varphi(b_i) = \psi(L_f \setminus \{\varphi(b_j) \mid j < i\}, b_i)$,
(P3) $K_f \supseteq \psi(L_f, B_f)$.

Indeed, (P1) and (P2) follow from the construction, and (P3) follows from (P2). (P3) immediately implies that K is an ε-kernel of P. Next, we describe the procedures for updating K_f when L_f changes. These procedures maintain (P1)–(P3), thereby ensuring that the algorithm maintains an ε-kernel.

Inserting a point. Suppose a point p is inserted into L_f. We add p to X. Suppose $p \in \mathbb{H}_J$. We recompute $\psi(X_J, B_f)$. Next, we update $\varphi(\cdot)$ and K as follows. We maintain a point $\xi \in L_f$. Initially, ξ is set to p. Suppose we have processed $b_1, \ldots, b_{i-1}$. Let $\eta \in L_f$ be the current $\varphi(b_i)$. If $\|\hat{\xi} - b_i\| \leq \|\hat{\eta} - b_i\|$, then we swap ξ and $\varphi(b_i)$, otherwise neither ξ nor $\varphi(b_i)$ is updated. We then process b_{i+1}. After processing all points of B_f if $\xi = p$, i.e., no $\varphi(b_i)$ is updated, we stop. Otherwise, we add p to K_f and delete ξ from K_f. The insertion procedure makes at most two changes in K_f, and it can be verified that (P1)-(P3) are maintained.

Deleting a point. Suppose p is deleted from L_f. Suppose $p \in \mathbb{H}_J$. If $p \notin K_f$, then $p \in X$. We delete p from X and X_J and recompute $\psi(X_J, B)$. If $p \in K_f$, i.e., there is a $b_i \in B$ with $p = \varphi(b_i)$, then $p \notin X$. We delete p from K_f and K, recompute $\varphi(b_i)$, and add the new $\varphi(b_i)$ to K_f. Let $\varphi(b_i) \in \mathbb{H}_J$; we remove $\varphi(b_i)$ from X_J and recompute $\psi(X_J, B_f)$. We modify the ordering of B_f by moving b_i from its current position to the end. This is the only place where the ordering of B_f is modified. Since b_i is now the last point in the ordering of B_f, the new $\varphi(b_i)$ does not affect any other $\varphi(b_j)$. The deletion procedure also makes at most two changes in K_f and maintains (P1)–(P3).

Finally, insertion or deletion of a point in P causes at most one insertion plus one deletion in L_f, therefore we can conclude the following:

Lemma 6. *Let P be a set of n points in $\mathbb{R}^d$, A a set of anchor points of P, and $0 < \varepsilon < 1$ a parameter. P can be preprocessed in $O(n + 1/\varepsilon^{d-1})$ time into a data structure so that a stable ε-kernel of P with respect to A of size $O(1/\varepsilon^{(d-1)/2})$ can be maintained in $O(1/\varepsilon^{(d-1)/2})$ time under insertion and deletion, provided that A remains an anchor set of P.*

Updating anchors. We now describe the algorithm for maintaining a stable ε-kernel when anchors of P are no longer fixed and need to be updated dynamically. Roughly speaking, we divide P into *inner* and *outer* subsets of points. The outer subset acts as a *shield* so that a stable kernel of the inner subset with respect to a fixed anchor can be maintained using Lemma 4 or 6. When the outer subset can no longer act as a shield, we reconstruct the inner and outer sets and start the algorithm again. We refer to the duration between two consecutive reconstruction steps as an *epoch.* The algorithm maintains a stable kernel within each epoch, and the amortized number of changes in the kernel because of reconstruction at the beginning of a new epoch will be $O(1)$. We can use a de-amortization technique to make the ε-kernel stable across epochs. We now describe the algorithm in detail.

In the beginning of each epoch, we perform the following preprocessing. Set $\alpha = 1/10$ and compute a α-kernel $\mathcal{L}$ of P of size $O(\log n)$ using Chan's dynamic algorithm; we do not need the stable version of his algorithm advertised above. $\mathcal{L}$ can be updated in $O(\log n)$ time per insertion/deletion. We choose a parameter m, which is set to $1/\varepsilon^{d-1}$ or $1/\varepsilon^{(d-1)/2}$. We create the outer subset of P by peeling off m "layers" of anchor points $A_1, \ldots, A_m$. Initially, we set $P_0 = P$. Suppose we have constructed $A_0, \ldots, A_{i-1}$. Set $P_{i-1} = P \setminus \bigcup_{j=1}^{i-1} A_j$, and $\mathcal{L}$ is an α-kernel of P_{i-1}. Next, we construct the anchor set A_i of $\mathcal{L}$ as described earlier in this section. We set $P_i = P_{i-1} \setminus A_i$ and update $\mathcal{L}$ so that it is an α-kernel of P_i. Let $\mathcal{A} = \bigcup_i A_i$, $A = A_m$, and $P_I = P \setminus \mathcal{A}$. Let $\mathbb{H} = (1+\alpha)I_A$. By construction $P_I \subset \mathbb{H}$. $\mathcal{A}$ forms the outer subset and acts as a shield for P_I, which is the inner subset. Set $\delta = \varepsilon/(2(1+\alpha)(\beta_d)^2)$, where β_d is the constant in Lemma 3.

If $m = 1/\varepsilon^{d-1}$ (resp. $1/\varepsilon^{(d-1)/2}$), we maintain a stable δ-kernel K_I of P_I with respect to A of size $O(m)$ using Lemma 4 (resp. Lemma 6). Set $K = K_I \cup \mathcal{A}$; $|K| = O(m)$. We prove below that K is an ε-kernel of P. Let p be a point that is inserted into or deleted from P. If $p \in \mathbb{H}$, then we update K_I using Lemma 4 or 6. On the other hand, if p lies outside $\mathbb{H}$, we insert it into or delete it from $\mathcal{A}$. Once $\mathcal{A}$ has been updated m times, we end the current epoch and discard the current K. We begin a new epoch and reconstruct $\mathcal{A}$, P_I, and K_I as described above.

The preprocessing step at the beginning of a new epoch causes $O(m)$ changes in K and there are at least m updates in each epoch, therefore the algorithm maintains a stable kernel in the amortized sense. Again, using a de-amortization

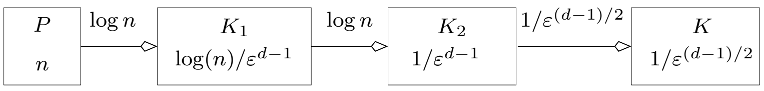

P
n
$\log n$
K_1
$\log(n)/\varepsilon^{d-1}$
$\log n$
K_2
$1/\varepsilon^{d-1}$
$1/\varepsilon^{(d-1)/2}$
K
$1/\varepsilon^{(d-1)/2}$

Fig. 2. Composing stable ε-kernel algorithms

technique, we can ensure that K is stable. The correctness of the algorithm follows from the following lemma (proved in full version [3]).

Lemma 7. *K is always an ε-kernel of P.*

Using Lemmas 4 and 6, we can bound the amortized update time and conclude the following.

Lemma 8. *For a set P of n points in $\mathbb{R}^d$ and a parameter $0 < \varepsilon < 1$, there is a data structure that can maintain a stable ε-kernel of P of size $O(1/\varepsilon^{(d-1)/2})$ under insertions and deletions in amortized time $O(n\varepsilon^{(d-1)/2} + 1/\varepsilon^{(d-1)/2} + \log n)$, or of size $O(1/\varepsilon^{d-1})$ in amortized time $O(n\varepsilon^{d-1} + \log n + \log(1/\varepsilon))$.*

Putting it together. For a point set $P \subset \mathbb{R}^d$ of size n, we can produce the best size and update time tradeoff for stable ε-kernels by invoking Lemma 2 to compose three stable ε-kernel algorithms, as illustrated in Figure 2. We first apply Lemma 1 to maintain a stable $(\varepsilon/3)$-kernel K_1 of P of size $O(\min\{n, (1/\varepsilon^{d-1}) \cdot \log n\})$ with update time $O(\log n)$. We then apply Lemma 8 to maintain a stable $(\varepsilon/3)$-kernel K_2 of K_1 of size $O(1/\varepsilon^{d-1})$ with update time $O(|K_1|\varepsilon^{d-1} + \log|K_1| + \log(1/\varepsilon)) = O(\log n + \log(1/\varepsilon))$. Finally we apply Lemma 8 again to maintain a stable $(\varepsilon/3)$-kernel K of K_2 of size $O(1/\varepsilon^{(d-1)/2})$ with update time $O(|K_2|\varepsilon^{(d-1)/2} + 1/\varepsilon^{(d-1)/2} + \log|K_2|) = O(1/\varepsilon^{(d-1)/2})$. K is a stable ε-kernel of P of size $O(1/\varepsilon^{(d-1)/2})$ with update time $O(\log n + 1/\varepsilon^{(d-1)/2})$. This completes the proof of Theorem 1.

3 Approximation Stability

In this section we prove the upper bound in Theorem 2. Due to lack of space we only prove the upper bound for $d = 2, 3$; the remainder is in the full version [3].

By [1], it suffices to consider the case in which P is fat and the diameter of P is normalized to 1. Let K be an ε-kernel of P of the smallest size. Let $\mathcal{P} = \mathrm{conv}(K)$, and $\mathcal{P}_\varepsilon = \mathcal{P} \oplus \varepsilon\mathbb{B}^d$. We have $\mathcal{P} \subseteq \mathrm{conv}(P) \subseteq \mathcal{P}_\varepsilon$ by the definition of ε-kernels. It suffices to show that there is a set $K' \subseteq P$ such that for $\mathcal{P}' = \mathrm{conv}(K')$, $\mathcal{P}' \subseteq \mathrm{conv}(P) \subseteq \mathcal{P}'_{\varepsilon/2}$, and $|K'| = O(|K|^{\lfloor d/2 \rfloor} \log^{d-2}(1/\varepsilon))$ [1].

For convenience, we assume that K' is not necessarily a subset of points in P; instead, we only require K' to be a subset of points in $\mathrm{conv}(P)$. By Caratheodory's theorem, for each point $x \in K$, we can choose a set $P_x \subseteq P$ of at most $d + 1$ points such that $x \in \mathrm{conv}(P_x)$. We set $\bigcup_{x \in K'} P_x$ as the desired $(\varepsilon/2)$-kernel of P; $|\bigcup_{x \in K'} P_x| \le (d + 1)|K'| = O(\kappa(P, \varepsilon)^{\lfloor d/2 \rfloor} \log^{d-2}(1/\varepsilon))$.

Initially, we add a point into K' for each point in K. If $p \in K$ lies on $\partial\,\mathrm{conv}(P)$, we add p to K'. Otherwise we project p onto $\partial\,\mathrm{conv}(P)$ in a direction in which p is maximal in K and add the projected point to K'. Abusing the notation slightly, we use $\mathcal{P}$ to denote hull of these initial points. For simplicity, we assume $\mathcal{P}$ to be a simplicial polytope.

Decomposition of $\mathcal{P}_\varepsilon \setminus \operatorname{intr} \mathcal{P}$. There are d types of simplices on $\partial\mathcal{P}$. In $\mathbb{R}^2$ these are points and edges. In $\mathbb{R}^3$ these are points, edges, and triangles. We can decompose $\mathcal{P}_\varepsilon \setminus \operatorname{intr} \mathcal{P}$ into a set of regions, each region $\sigma(f)$ corresponding to a simplex f in $\mathcal{P}$. For each simplex f in $\mathcal{P}$ let $f^* \subseteq \mathbb{S}^{d-1}$ denote the dual of f in the Gaussian diagram of $\mathcal{P}$. Recall that if f has dimension k ($0 \le k \le d-1$), then f^* has dimension $d-1-k$. The region $\mathcal{P}_\varepsilon \setminus \operatorname{intr} \mathcal{P}$ is partitioned into a collection of $|\mathcal{P}|$ regions (where $|\mathcal{P}|$ is the number of faces of all dimensions in $\mathcal{P}$). Each simplex f in $\mathcal{P}$ corresponds to a region defined

$$\sigma(f) = \{f + zu \mid 0 \le z \le \varepsilon,\ u \in f^*\}.$$

For a subsimplex $\tau \in f$, we can similarly define a region $\sigma(\tau) = \{\tau + zu \mid 0 \le z \le \varepsilon,\ u \in f^*\}$. In $\mathbb{R}^2$, there are two types of regions: point regions and edge regions. In $\mathbb{R}^3$, there are three types of regions: point regions (see Figure 4(a)), edge regions (see Figure 4(b)), and triangle regions (see Figure 4(c)).

For convenience, for any point $q = \bar{q} + z \cdot u \in \sigma(f)$, where $\bar{q} \in f, 0 \le z \le \varepsilon$, and $u \in f^*$, we write $q = \bar{q}[u, z]$ (which intuitively reads, the point whose projection onto f is $\bar{q}$ and which is at a distance z above f in direction u). We also write $q[v] = \bar{q} + z \cdot v$ (intuitively, $q[v]$ is obtained by rotating q w.r.t. f from direction u to direction v). Similarly, we write a simplex $\bar{\Delta}[u, z] = \bar{\Delta} \oplus z \cdot u$, where $\bar{\Delta}$ is a simplex inside f, $0 \le z \le \varepsilon$, and $u \in f^*$, and write $\Delta[v] = \bar{\Delta} \oplus z \cdot v$.

We will proceed to prove the upper bound as follows. For each type of region $\sigma(f)$ we place a bounded number of points from $\sigma(f) \cap \operatorname{conv}(P)$ into K' and then prove that all points in $\sigma(f) \cap \operatorname{conv}(P)$ are within a distance $\varepsilon/2$ from some point in $\mathcal{P}' = \operatorname{conv}(K')$. We begin by introducing three ways of "gridding" $\sigma(f)$ and then use these techniques to directly prove results for several base cases, which illustrate the main conceptual ideas. These base cases will already be enough to prove the results in $\mathbb{R}^2$ and $\mathbb{R}^3$. In the full version [3] we generalize this to $\mathbb{R}^d$ using an involved recursive construction. We set a few global values: $\delta = \varepsilon/12d$, $\theta = 2\arcsin(\delta/2\varepsilon)$, and $\rho = \delta/\varepsilon$.

1: Creating layers. For a point $q = \bar{q}[u, z] \in \sigma(f)$ we classify it depending on the value $z = |q - \bar{q}|$. If $z \le \varepsilon/2$, then q is already within $\varepsilon/2$ of $\mathcal{P}$. We then divide the range $[\varepsilon/2, \varepsilon]$ into a constant $H = (\varepsilon/2)/\delta$ number of cases using $\mathcal{H} = \{h_1 = \varepsilon/2, h_2 = \varepsilon/2 + \delta, \ldots, h_H = \varepsilon - \delta\}$. If $z \in [h_i, h_{i+1})$, then we set $q_{h_i} = \bar{q}[u, h_i]$. We define $\Psi_{f,h_i} \subset \sigma(f) \cap \operatorname{conv}(P)$ to be the set of points that are a distance exactly h_i from f.

2: Discretize angles. We create a constant size θ-net $U_{f,h} = \{u_1, u_2, \ldots\} \subset f^*$ of directions with the following properties. (1) For each $q = \bar{q}[u, h] \in \Psi_{f,h}$ there is a direction $u_i \in U_{f,h}$ such that the angle between u and u_i is at most θ. (2) For each $u_i \in U_{f,h}$ there is a point $p_i = \bar{p}_i[u_i, h] \in \Psi_{f,h}$; let $N_{f,h} = \{p_i \mid i \ge 1\}$. $U_{f,h}$ is constructed by first taking a $(\theta/2)$-net U_f of f^*, then for each $u_i' \in U_f$ choosing a point $p_i = \bar{q}_i[u_i, h] \in \Psi_{f,h}$ where u_i is within an angle $\theta/2$ of u_i' (if one exists), and finally placing u_i in $U_{f,h}$.

3: Exponential grid. Define a set $\mathcal{D} = \{d_0, d_1 = (1+\rho)d_0, \ldots, d_m = (1+\rho)^m d_0\}$ of distances where $d_m < 1$ and $d_0 = \delta$, so $m = O(\log 1/\varepsilon)$. For a face $f \in \mathcal{P}$, let any $r \in \sigma(f)$ be called a *support point of* f. Let $p_1, \ldots, p_k$ be the vertices of the k-simplex f. For each p_j, and each $d_i \in \mathcal{D}$ (where $d_i < ||p_j - \bar{r}||$), let $p_{j,i}$ be the point at distance d_i from p_j on the segment $p_j\bar{r}$. For each boundary facet F of f, define a sequence of at most m simplices $F_0, F_1, \ldots \in \operatorname{conv}(F \cup \bar{r})$, each a homothet of F, so the vertices of F_i lie on segments $p_j\bar{r}$ where $p_j \in \partial F$ (see Figure 5(a)). The translation of each F_i is defined so it intersects a point $p_{j,i}$ (where $p_j \in \partial F$) and is as close to F as possible. This set of $(k-1)$-simplices for each F defines the exponential grid $G_{r,f}$. The full grid structure is revealed as this is applied recursively on each F_i.

The exponential grid $G_{r,\Delta}$ on a simplex Δ has two important properties for a point $q \in \Delta$:

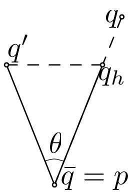
(G1) If $q \in \operatorname{conv}(F \cap \bar{r})$ lies between boundary facet F and F_0, let q_0 be the intersection of the line segment $q\bar{r}$ with F_0; then $||q - q_0|| \leq d_0 = \delta$.

(G2) If $q \in \operatorname{conv}(F \cap \bar{r})$ lies between F_{i-1} and F_i and the segment $q\bar{r}$ intersects F_i at q_i, let q_F be the intersection of F with the ray $\overrightarrow{rq}$; then $||q_i - q||/||q_i - q_F|| \leq \rho = \delta/\varepsilon$.

We now describe how to handle certain simple types of regions: where f is a point or an edge. These will be handled the same regardless of the dimension of the problem, and they (the edge case in particular) will be used as important base cases for higher dimensional problems.

Point regions. Consider a point region $\sigma(p)$. For each $h \in \mathcal{H}$ create θ-net $U_{p,h}$ for $\Psi_{p,h}$, so $N_{p,h}$ are the corresponding points where each $p_i = p[h, u_i] \in N_{p,h}$ has $u_i \in U_{p,h}$. Put each $N_{p,h}$ in K'.

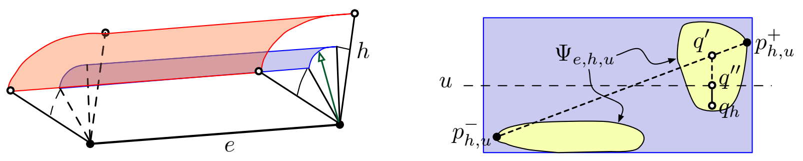
For any point $q = \bar{q}[u', z] \in \sigma(p) \cap \operatorname{conv}(P)$, let $q' = \bar{q}[u, h]$ where $h \in \mathcal{H}$ is the largest value such that $h \leq z$ and $u \in U_{p,h}$ is the closest direction to u'; set $q_h = \bar{q}[u', h] = q'[u']$. First $||q - q_h|| \leq \delta$ because $z - h \leq \delta$. Second $||q_h - q'|| \leq \delta$ because the angle between u' and u is at most θ, and they are rotated about the point p. Thus $||q - q'|| \leq ||q - q_h|| + ||q_h - q'|| \leq 2\delta \leq \varepsilon/2$.



Lemma 9. *For a point region* $\sigma(p)$*, there exists a constant number of points* $K_p \subset \sigma(p) \cap \operatorname{conv}(P)$ *such that all points* $q \in \sigma(p) \cap \operatorname{conv}(P)$ *are within a distance* $\varepsilon/2$ *of* $\operatorname{conv}(K_p)$.

Edge regions. Consider an edge region $\sigma(e)$ for an edge e of $\mathcal{P}$. Orient e along the x-axis. For each $h \in \mathcal{H}$ and $u \in U_{e,h}$, let $\Psi_{e,h,u}$ be the set of points in $\Psi_{e,h}$ within an angle θ of u. For each $\Psi_{e,h,u}$, we add to K_e the (two) points of $\Psi_{e,h,u}$ with the largest and smallest x-coordinates, denoted by $p^+_{h,u}$ and $p^-_{h,u}$.

For any point $q = \bar{q}[v, z] \in \sigma(e) \cap \operatorname{conv}(P)$, there is a point $q'' = \bar{q}[u, h]$ such that $h \in \mathcal{H}$ is the largest value less than z and $u \in U_{e,h}$ is the closest direction to v. Furthermore, $||q - q''|| \leq ||q - q_h|| + ||q_h - q''|| \leq (z-h) + 2\varepsilon\sin(\theta/2) = \delta + \delta = 2\delta$. We can also argue that there is a point $q' = \bar{q}[u', z'] \in p^-_{h,u}p^+_{h,u}$, because if $\bar{q}$ has



(a) $\sigma(e)$ in $\mathbb{R}^3$ (b) top view of $\sigma(e)$ at height h

Fig. 3. Illustration of 2 points in K' for edge case with specific $h \in \mathcal{H}$ and $u \in U_{e,h,\theta}$

smaller x-coordinate than $\bar{p}^-_{h,u}$ or larger x-coordinate than $\bar{p}^+_{h,u}$, then q' cannot be in $\Psi_{e,h,u}$. Clearly the angle between u and u' is less than θ. This also implies that $h - z' < \delta$. Thus $||q'' - q'|| \leq 2\delta$, implying $||q - q'|| \leq 4\delta \leq \varepsilon/2$.

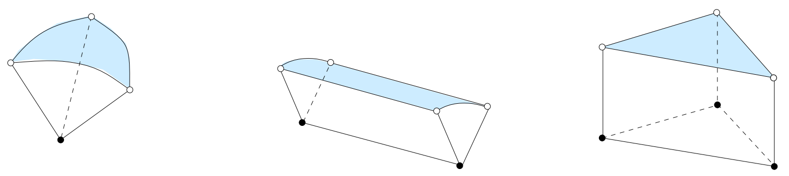
Lemma 10. *For an edge region $\sigma(e)$, there exists $O(1)$ points $K_e \subset \sigma(e) \cap \operatorname{conv}(P)$ such that for any point $q = \bar{q}[z, v] \in \sigma(e) \cap \operatorname{conv}(P)$ there is a point $p = \bar{q}[h, u] \in \operatorname{conv}(K_e)$ such that $z - h \leq 2\delta$, $||v - u|| \leq 2\delta$, and, in particular, $||q - p|| \leq 4\delta \leq \varepsilon/2$.*

For $K \subset P \in \mathbb{R}^2$ there are $|K|$ points and edges in $\mathcal{P}$. Thus combining Lemmas 9 and 10 $|K'|/|K| = O(1)$ and we have proven Theorem 2 for $d = 2$. Next, we prove the theorem for $d = 3$.

Construction of K'. Now consider $K \subset P \in \mathbb{R}^3$ and the point regions, edge regions, and triangle regions in the decomposition of $\mathcal{P}_\varepsilon \setminus \operatorname{intr} \mathcal{P}$ (see Figure 4). By Lemmas 9 and 10 we can add $O(|K|)$ points to K' to account for all point and edge regions. We can now focus on the $O(|K|)$ triangle regions.

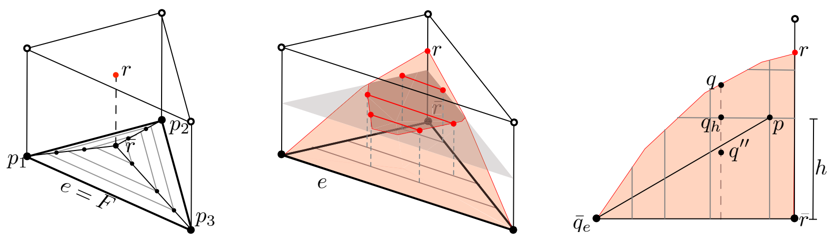
Consider a triangle region $\sigma(t)$ for a triangle t in $\mathcal{P}$ (see Figure 5(a)), t^* consists of a single direction, the one normal to t. Let r be the highest point of $\sigma(t) \cap \operatorname{conv}(P)$ in direction t^*. We add r to K' and we create an exponential grid $G_{r,t}$ with r as the support point. For each edge $e \in G_{r,t}$ and $h \in \mathcal{H}$ we add the intersection of $e[t^*, h]$ with the boundary of $\sigma(t) \cap \operatorname{conv}(P)$ to K', as shown in Figure 5(b). Thus, in total we add $O(|K| \log(1/\varepsilon))$ points to K'.

Proof of correctness. Consider any point $q = \bar{q}[t^*, z] \in \sigma(t) \cap \operatorname{conv}(P)$ and associate it with a boundary edge e of t such that $\bar{q} \in \operatorname{conv}(e \cup \bar{r})$. Let $q_h = \bar{q}[t^*, h]$ where $h \in \mathcal{H}$ is the largest height such that $h \leq z$. If segment $\bar{q}\bar{r}$ does not



(a) f is a vertex of $\mathcal{P}$ (b) f is an edge of $\mathcal{P}$ (c) f is a facet of $\mathcal{P}$

Fig. 4. Illustration of regions in the partition of $\mathcal{P}_\varepsilon \setminus \operatorname{intr} \mathcal{P}$ in three dimensions



(a) $\sigma(t)$ with r and $G_{r,t}$ (b) Subtriangle t_e of $\sigma(t)$ (c) Slice of (b) through r, q

Fig. 5. Illustration to aid correctness of approximation of triangle regions in $\mathbb{R}^3$

intersect any edge e_i parallel to e in $G_{r,t}$, let $\bar{p} = \bar{r}$. Otherwise, let e_i be the first segment parallel to e in $G_{r,t}$ intersected by the ray $\overrightarrow{\bar{q}\bar{r}}$, and let $\bar{p}$ be the intersection. Let $p = \bar{p}[t^*, h]$ which must be in $\mathrm{conv}(K')$ by construction. If $e_i = e_0$, then by (G1) we have $||q_h - p|| = ||\bar{q} - \bar{p}|| \leq \delta$, thus $||q - p|| \leq 2\delta \leq \varepsilon/2$ and we are done. Otherwise, let $\bar{q}_e$ be the intersection of e with ray $\overrightarrow{\bar{r}\bar{q}}$. By (G2) $||\bar{p} - \bar{q}||/||\bar{p} - \bar{q}_e|| \leq \rho = \delta/\varepsilon$. Thus, $q'' = \bar{q}[t^*, h - \varepsilon\rho]$ is below the segment $\bar{q}_e p$ (see Figure 5(c)) and thus $q'' \in \mathrm{conv}(K')$ since triangle $p\bar{p}\bar{q}_e$ is in $\mathrm{conv}(K')$. Finally, $||q - q''|| = ||q - q_h|| + ||q_h - q''|| \leq 2\delta \leq \varepsilon/2$. This proves Theorem 2 for $d = 3$.

3.1 Remarks

(1) For $d = 2, 3$, $\kappa(P, \varepsilon/2)$ is only a factor of $O(1)$ and $O(\log(1/\varepsilon))$, respectively, larger than $\kappa(P, \varepsilon)$; therefore, the sizes of optimal ε-kernels in these dimensions are relative stable. However, for $d \geq 4$, the stability drastically reduces in the worst case because of the superlinear dependency on $\kappa(P, \varepsilon)$.

(2) Neither the upper nor the lower bound in the theorem is tight. For $d = 3$, we can prove a tighter lower bound of $\Omega\big(\kappa(P, \varepsilon)\log(1/(\varepsilon \cdot \kappa(P, \varepsilon)))\big)$. We conjecture in $\mathbb{R}^d$ that

$$\kappa(P, \varepsilon/2) = \Theta\big(\kappa(P, \varepsilon)^{\lfloor d/2 \rfloor} \log^{d-2}(1/(\varepsilon^{(d-1)/2} \cdot \kappa(P, \varepsilon)))\big).$$

References

1. Agarwal, P.K., Har-Peled, S., Varadarajan, K.: Approximating extent measure of points. Journal of ACM 51(4), 606–635 (2004)
2. Agarwal, P.K., Har-Peled, S., Varadarajan, K.: Geometric approximations via coresets. In: Combinatorial and Computational Geometry, pp. 1–31 (2005)
3. Agarwal, P.K., Phillips, J.M., Yu, H.: Stability of ε-kernels. arXiv:1003.5874
4. Agarwal, P.K., Yu, H.: A space-optimal data-stream algorithm for coresets in the plane. In: SoCG, pp. 1–10 (2007)
5. Barequet, G., Har-Peled, S.: Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. Journ. of Algs. 38, 91–109 (2001)
6. Chan, T.: Faster core-set constructions and data-stream algorithms in fixed dimensions. Computational Geometry: Theory and Applications 35, 20–35 (2006)
7. Chan, T.: Dynamic coresets. In: SoCG, pp. 1–9 (2008)

8. Har-Peled, S.: Approximation Algorithm in Geometry, ch. 22 (2010), `http://valis.cs.uiuc.edu/~sariel/teach/notes/aprx/`
9. Hershberger, J., Suri, S.: Adaptive sampling for geometric problems over data streams. Computational Geometry: Theory and Applications 39, 191–208 (2008)
10. Yu, H., Agarwal, P.K., Poreddy, R., Varadarajan, K.: Practical methods for shape fitting and kinetic data structures using coresets. Algorithmica 52, 378–402 (2008)
11. Zarrabi-Zadeh, H.: An almost space-optimal streaming algorithm for coresets in fixed dimensions. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 817–829. Springer, Heidelberg (2008)

The Geodesic Diameter of Polygonal Domains*

Sang Won Bae[1], Matias Korman[2], and Yoshio Okamoto[3]

[1] Department of Computer Science, Kyonggi University, Korea
swbae@kgu.ac.kr

[2] Computer Science Department, Université Libre de Bruxelles (ULB), Belgium
mkormanc@ulb.ac.be

[3] Graduate School of Information Science and Engineering, Tokyo Institute of Technology, Tokyo, Japan
okamoto@is.titech.ac.jp

Abstract. This paper studies the geodesic diameter of polygonal domains having h holes and n corners. For simple polygons (i.e., $h = 0$), it is known that the geodesic diameter is determined by a pair of corners of a given polygon and can be computed in linear time. For general polygonal domains with $h \geq 1$, however, no algorithm for computing the geodesic diameter was known prior to this paper. In this paper, we present the first algorithm that computes the geodesic diameter of a given polygonal domain in worst-case time $O(n^{7.73})$ or $O(n^7(\log n + h))$. Among other results, we show the following geometric observation: the geodesic diameter can be determined by two points in its interior. In such a case, there are at least five shortest paths between the points.

1 Introduction

In this paper, we address the geodesic diameter problem in polygonal domains. Intuitively, given a polygonal domain $\mathcal{P}$ with holes, the geodesic distance $\mathrm{d}(p, q)$ between points p and q of $\mathcal{P}$ is defined as the length of a shortest path between them, among all the paths that stay within $\mathcal{P}$. The geodesic diameter $\mathrm{diam}(\mathcal{P})$ is defined as the largest geodesic distance between any two points of $\mathcal{P}$.

For simple polygons (i.e., domains with no holes), the geodesic diameter has been extensively studied. Chazelle [7] provided the first $O(n^2)$-time algorithm computing the geodesic diameter of a simple polygon, where n is the number of corners of $\mathcal{P}$. Afterwards, Suri [19] presented an $O(n \log n)$-time algorithm that solves the all-geodesic-farthest neighbors problem, computing the farthest neighbor of every corner and thus finding the geodesic diameter. At last, Hershberger and Suri [12] showed that the diameter can be computed in linear time using their fast matrix search technique.

* Work by S.W. Bae was supported by National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0005974). Work by Y. Okamoto was supported by Global COE Program "Computationism as a Foundation for the Sciences" and Grant-in-Aid for Scientific Research from Ministry of Education, Science and Culture, Japan, and Japan Society for the Promotion of Science.

On the other hand, the geodesic diameter of a domain with holes is less understood. Mitchell [15] has posed an open problem asking an algorithm for computing the geodesic diameter of a polygonal domain. Moreover, even for the corner-to-corner diameter $\max_{u,v \in V} \mathrm{d}(u, v)$, we know nothing but the brute-force algorithm that takes $O(n^2 \log n)$ time, checking all the geodesic distances between every pair of corners.[1]

This fairly wide gap between simple polygons and polygonal domains is seemingly due to the uniqueness of the shortest path between any two points. When no holes exist, it is well known that there is a unique shortest path between any two points [10]. Using this uniqueness, one can show that the diameter is realized by a pair of corners [12, 19]. For general polygonal domains, however, this is not the case. In this paper, we exhibit several examples where the diameter is realized by non-corner points on $\partial\mathcal{P}$ or even by interior points of $\mathcal{P}$. (See Fig. 1.) This observation also shows an immediate difficulty in devising any exhaustive algorithm since the search space is not discrete.

The status of the geodesic center problem is also similar. A point in $\mathcal{P}$ is defined as a *geodesic center* if it minimizes the maximum geodesic distance from it to any other point of $\mathcal{P}$. Asano and Toussaint [3] introduced the first $O(n^4 \log n)$-time algorithm for computing the geodesic center of a simple polygon (i.e., when $h = 0$), and Pollack, Sharir and Rote [18] improved it to $O(n \log n)$ time. As with the diameter problem, there is no known algorithm for domains with holes. See O'Rourke and Suri [17] and Mitchell [15] for more references on the geodesic diameter/center problem.

Since the geodesic diameter/center of a simple polygon is determined by its corners, one can exploit the *geodesic farthest-site Voronoi diagram* of the corners V to compute the diameter/center, which can be built in $O(n \log n)$ time [2]. Recently, Bae and Chwa [4] presented an $O(nk \log^3(n + k))$-time algorithm for computing the geodesic farthest-site Voronoi diagram of k sites in polygonal domains with holes. This result can be used to compute the geodesic diameter $\max_{p,q \in S} \mathrm{d}(p, q)$ of a *finite* set S of points in $\mathcal{P}$. However, this approach cannot be directly used for computing $\mathrm{diam}(\mathcal{P})$ without any characterization of the diameter. Moreover, when $S = V$, this approach is no better than the brute-force $O(n^2 \log n)$-time algorithm for computing the corner-to-corner diameter $\max_{u,v \in V} \mathrm{d}(u, v)$.

In this paper, we present the first algorithms that compute the geodesic diameter of a given polygonal domain in $O(n^{7.73})$ or $O(n^7(\log n + h))$ time in the worst case. Our new geometric results underlying the algorithms show that the existence of any diametral pair consisting of non-corner points implies multiple shortest paths between the pair; Among other results, we show that *if (s, t) is a diametral pair and both s and t lie in the interior of $\mathcal{P}$, then there are at least five shortest paths between s and t.*

Some analogies between polygonal domains and convex polytopes in $\mathbb{R}^3$ can be seen. O'Rourke and Schevon [16] proved that if the geodesic diameter on a convex 3-polytope is realized by two non-corner points, at least five shortest

[1] Personal communication with Mitchell.

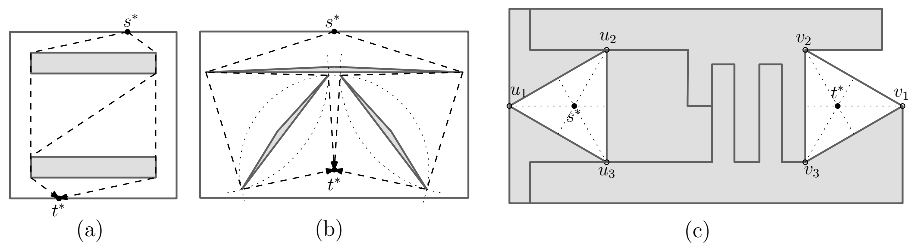


Fig. 1. Three polygonal domains where the geodesic diameter is determined by a pair (s^*, t^*) of non-corner points; Gray-shaded regions depict the interior of the holes and dark gray segments depict the boundary $\partial\mathcal{P}$. Recall that $\mathcal{P}$, as a set, contains its boundary $\partial\mathcal{P}$. (a) Both s^* and t^* lie on $\partial\mathcal{P}$. There are three shortest paths between s^* and t^*. In this polygonal domain, there are two (symmetric) diametral pairs. (b) $s^* \in \partial\mathcal{P}\setminus V$ and $t^* \in \mathrm{int}\mathcal{P}$. Three triangular holes are placed in a symmetric way. There are four shortest paths between s^* and t^*. (c) Both s^* and t^* lie in the interior $\mathrm{int}\mathcal{P}$. Here, the five holes are packed like jigsaw puzzle pieces, forming narrow corridors (dark gray paths) and two empty, regular triangles. Observe that $\mathrm{d}(u_1, v_1) = \mathrm{d}(u_1, v_2) = \mathrm{d}(u_2, v_2) = \mathrm{d}(u_2, v_3) = \mathrm{d}(u_3, v_3) = \mathrm{d}(u_3, v_1)$. The points s^* and t^* lie at the centers of the triangles formed by the u_i and the v_i, respectively. There are six shortest paths between s^* and t^*.

paths exist between the two (a simpler proof of the same fact was later shown by Zalgaller [20]). Based on this observation, they presented an $O(n^{14} \log n)$-time algorithm for computing the geodesic diameter on a convex 3-polytope. Afterwards, the time bound was improved to $O(n^8 \log n)$ by Agarwal et al. [1] and recently to $O(n^7 \log n)$ by Cook IV and Wenk [9]. Although our algorithms seemingly require fairly large amount of time, notice that they are comparable with the runtimes for convex polytopes.

Due to the space constraints, some of the discussions and examples are omitted in this version. Details can be found in the arXiv version [5].

2 Preliminaries

A *polygonal domain* $\mathcal{P}$ with h holes and n corners V is a connected and closed subset of $\mathbb{R}^2$ with h holes whose boundary $\partial\mathcal{P}$ consists of $h + 1$ simple closed polygonal chains of n total line segments. The holes and the outer boundary of $\mathcal{P}$ are regarded as *obstacles* so that any feasible path in $\mathcal{P}$ is not allowed to cross the boundary $\partial\mathcal{P}$. The geodesic distance $\mathrm{d}(p, q)$ between any two points p, q in a polygonal domain $\mathcal{P}$ is defined as the length of a shortest obstacle-avoiding path between p and q, where the *length* of a path is the sum of the Euclidean lengths of its segments. It is well known from earlier work that there always exists a *shortest feasible path* between any two points $p, q \in \mathcal{P}$ [14]. A pair (s, t) of points in $\mathcal{P}$ that realizes the geodesic diameter $\mathrm{diam}(\mathcal{P})$ is called a *diametral pair*.

Throughout the paper, we frequently use several topological concepts such as open and closed subsets, neighborhoods, and the boundary ∂A and the interior int$A$ of a set A; all of them are derived with respect to the standard topology on $\mathbb{R}^d$ with the Euclidean norm $\|\cdot\|$ for fixed $d \geq 1$. We call a k-dimensional affine subspace of $\mathbb{R}^d$ a *k-flat* and denote the straight line segment joining two points a, b by $\overline{ab}$.

Let V be the set of all corners of $\mathcal{P}$ and $\pi(s,t)$ be a shortest path between $s \in \mathcal{P}$ and $t \in \mathcal{P}$. Such a path is represented as a sequence $\pi(s,t) = (s, v_1, \ldots, v_k, t)$ for some $v_1, \ldots, v_k \in V$; that is, a polygonal chain through a sequence of corners [14]. Note that we may have $k = 0$ when $\mathrm{d}(s,t) = \|s - t\|$. If two paths (with possibly different endpoints) induce the same sequence of corners, then they are said to have the same *combinatorial structure*.

The *shortest path map* $\mathsf{SPM}(s)$ for a fixed $s \in \mathcal{P}$ is a decomposition of $\mathcal{P}$ into cells such that every point in a common cell can be reached from s by shortest paths of the same combinatorial structure. Each cell $\sigma_s(v)$ of $\mathsf{SPM}(s)$ is associated with a corner $v \in V$ which is the last corner of $\pi(s,t)$ for any t in the cell $\sigma_s(v)$. We also define the cell $\sigma_s(s)$ as the set of points $t \in \mathcal{P}$ in which $\pi(s,t)$ passes through no corner of $\mathcal{P}$. Each edge of $\mathsf{SPM}(s)$ is an arc on the boundary of two incident cells $\sigma_s(v_1)$ and $\sigma_s(v_2)$ determined by two corners $v_1, v_2 \in V \cup \{s\}$. Similarly, each vertex of $\mathsf{SPM}(s)$ is determined by at least three corners $v_1, v_2, v_3 \in V \cup \{s\}$. Note that for fixed $s \in \mathcal{P}$ a point whose distance to s is the largest lies at either (1) a vertex of $\mathsf{SPM}(s)$, (2) an intersection between the boundary $\partial\mathcal{P}$ and an edge of $\mathsf{SPM}(s)$, or (3) a corner in V (otherwise a farther point can be found). The shortest path map $\mathsf{SPM}(s)$ has $O(n)$ complexity and can be computed in $O(n \log n)$ time using $O(n \log n)$ working space [13]. For more details on shortest path maps, see [14, 13, 15].

Path-length function. If $\pi(s,t) \neq \overline{st}$, let u and v be the first and last corners of V visited in $\pi(s,t)$, respectively. The path $\pi(s,t)$ is formed as the union of $\overline{su}$, $\overline{vt}$ and $\pi(u,v)$. Note that u and v are not necessarily distinct. In order to realize such a path, we assert that s is visible from u and t is visible from v. That is, $s \in \mathsf{VR}(u)$ and $t \in \mathsf{VR}(v)$, where $\mathsf{VR}(p)$ for any $p \in \mathcal{P}$ is defined to be the set of all points $q \in \mathcal{P}$ such that $\overline{pq} \subset \mathcal{P}$. The set $\mathsf{VR}(p)$ is called the *visibility region* of $p \in \mathcal{P}$ [8].

We now define the *path-length function* $\mathrm{len}_{u,v} \colon \mathsf{VR}(u) \times \mathsf{VR}(v) \to \mathbb{R}$ for any fixed pair of corners $u, v \in V$ to be $\mathrm{len}_{u,v}(s,t) := \|s - u\| + \mathrm{d}(u,v) + \|v - t\|$. That is, $\mathrm{len}_{u,v}(s,t)$ represents the length of the path from s to t that has the fixed combinatorial structure, entering u from s and exiting v to t. Also, unless $\mathrm{d}(s,t) = \|s - t\|$ (equivalently, $s \in \mathsf{VR}(t)$), the geodesic distance $\mathrm{d}(s,t)$ can be expressed as the pointwise minimum of some path-length functions:

$$\mathrm{d}(s,t) = \min_{u \in \mathsf{VR}(s),\ v \in \mathsf{VR}(t)} \mathrm{len}_{u,v}(s,t).$$

Consequently, we have two possibilities for a diametral pair (s^*, t^*); either we have $\mathrm{d}(s^*, t^*) = \|s^* - t^*\|$ or the pair (s^*, t^*) is a local maximum of the lower envelope of several path-length functions. We will mainly study the latter case, since the former can be easily handled.

2.1 Local Maxima of the Lower Envelope of Convex Functions

In this section, we give an interesting property of the lower envelope of a family of convex functions.

Theorem 1. *Let $\mathcal{F}$ be a finite family of real-valued convex functions defined on a convex subset $C \subseteq \mathbb{R}^d$ and $g(x) := \min_{f \in \mathcal{F}} f(x)$ be their lower envelope. Suppose that g attains a local maximum at an interior point $x^* \in C$ and there are exactly m functions $f_1, \ldots, f_m \in \mathcal{F}$ such that $m \leq d$ and $f_i(x^*) = g(x^*)$ for all $i = 1, \ldots, m$. If none of the f_i attains a local minimum at x^*, then there exists a $(d+1-m)$-flat $\varphi \subset \mathbb{R}^d$ through x^* such that g is constant on $\varphi \cap U$ for some neighborhood $U \subset \mathbb{R}^d$ of x^* with $U \subset C$.*

Here, we give a sketchy idea of our proof for the theorem. Consider a local maximum x^* of the lower envelope g of a family of functions. Let m be the number of functions $f \in \mathcal{F}$ such that $f(x^*) = g(x^*)$. Since g is the lower envelope, all other functions $f' \in \mathcal{F}$ must satisfy $f'(x^*) > g(x^*)$. In particular, the function g is the lower envelope of the m convex functions in a small neighborhood of x^*. Since no function attains a local minimum at x^*, for each function there exists at least one direction in which the function value does not decrease. We show something stronger: Using convexity of the f_i, we will show that there is a flat of dimension $d + 1 - m$ through x^* on which the m functions that define g must either increase or stay constant. Since x^* is a local maximum, the only possibility is that g remains constant in the flat.

3 Properties of Geodesic-Maximal Pairs

We call a pair $(s^*, t^*) \in \mathcal{P} \times \mathcal{P}$ *maximal* if (s^*, t^*) is a local maximum of the geodesic distance function d. That is, (s^*, t^*) is maximal if and only if there are two neighborhoods $U_s, U_t \subset \mathbb{R}^2$ of s^* and of t^*, respectively, such that for any $s \in U_s \cap \mathcal{P}$ and any $t \in U_t \cap \mathcal{P}$ we have $\mathrm{d}(s^*, t^*) \geq \mathrm{d}(s, t)$.

Let E be the set of all sides of $\mathcal{P}$ without their endpoints and $\mathcal{B}$ be their union. Note that $\mathcal{B} = \partial\mathcal{P} \setminus V$ is the boundary of $\mathcal{P}$ except the corners V. The goal of this section is to prove the following theorem, which is the main geometric result of this paper.

Theorem 2. *Suppose that (s^*, t^*) is a maximal pair in $\mathcal{P}$ that are not visible from each other, and $\Pi(s^*, t^*)$, V_{s^*}, and V_{t^*} are defined as above. We have the following implications.*

(V-V)	$s^* \in V$,	$t^* \in V$	*implies*	$\lvert\Pi(s^*, t^*)\rvert \geq 1, \lvert V_{s^*}\rvert \geq 1, \lvert V_{t^*}\rvert \geq 1;$
(V-B)	$s^* \in V$,	$t^* \in \mathcal{B}$	*implies*	$\lvert\Pi(s^*, t^*)\rvert \geq 2, \lvert V_{s^*}\rvert \geq 1, \lvert V_{t^*}\rvert \geq 2;$
(V-I)	$s^* \in V$,	$t^* \in \mathrm{int}\mathcal{P}$	*implies*	$\lvert\Pi(s^*, t^*)\rvert \geq 3, \lvert V_{s^*}\rvert \geq 1, \lvert V_{t^*}\rvert \geq 3;$
(B-B)	$s^* \in \mathcal{B}$,	$t^* \in \mathcal{B}$	*implies*	$\lvert\Pi(s^*, t^*)\rvert \geq 3, \lvert V_{s^*}\rvert \geq 2, \lvert V_{t^*}\rvert \geq 2;$
(B-I)	$s^* \in \mathcal{B}$,	$t^* \in \mathrm{int}\mathcal{P}$	*implies*	$\lvert\Pi(s^*, t^*)\rvert \geq 4, \lvert V_{s^*}\rvert \geq 2, \lvert V_{t^*}\rvert \geq 3;$
(I-I)	$s^* \in \mathrm{int}\mathcal{P}$,	$t^* \in \mathrm{int}\mathcal{P}$	*implies*	$\lvert\Pi(s^*, t^*)\rvert \geq 5, \lvert V_{s^*}\rvert \geq 3, \lvert V_{t^*}\rvert \geq 3.$

Moreover, each of the above bounds is tight.

We first give an idea of the proof. The general reasoning is roughly the same for all the different scenarios, and thus we focus on the case in which (s^*, t^*) is a maximal pair and both s^* and t^* are interior points. Consider the geodesic distance as a four-variate function in a neighborhood of (s^*, t^*). As mentioned in Section 2, the geodesic distance is the pointwise minimum of the family of path-length functions. Since the pair (s^*, t^*) is maximal, we want to apply Theorem 1. We will show that all the requirements are satisfied, which implies that the geodesic distance is constant in a flat of dimension $d + 1 - m = 5 - m$. However, we will also show that the geodesic distance function can only remain constant in a zero-dimensional flat (i.e., a point), hence $m \geq 5$. The main technical difficulty of the proof is the fact that the $\mathrm{len}_{u,v}$ are not globally defined. Thus, we must slightly redefine them in a way that all conditions of Theorem 1 are satisfied. In the other cases (boundary-interior, boundary-boundary, etc.) the boundary of $\mathcal{P}$ introduces additional constraints that reduce the degrees of freedom of the geodesic distance function. Hence, fewer paths are enough to pin the solution.

Throughout this section, for easy discussion, we assume that there is a *unique* shortest path between any two corners $u, v \in V$. This assumption does not affect Theorem 2 since multiple shortest paths between corners in V can only increase $|\Pi(s^*, t^*)|$. Note that this assumption implies that the pairs (u_i, v_i) are distinct, while the u_i (also the v_i) are not necessarily distinct. We then have $|V_{s^*}| \leq m$, $|V_{t^*}| \leq m$, and $|\{(u_i, v_i) \mid 1 \leq i \leq m\}| = m$, where $m = |\Pi(s^*, t^*)|$.

The following lemma proves the bounds on $|V_{s^*}|$ and $|V_{t^*}|$ of Theorem 2.

Lemma 1. *Let (s^*, t^*) be a maximal pair.*

1. *If $t^* \in \mathcal{B}$, then $|V_{t^*}| \geq 2$. Moreover, if $t^* \in e \in E$, then there exists $v \in V_{t^*}$ such that v is off the line supporting e.*
2. *If $t^* \in \mathrm{int}\mathcal{P}$, then $|V_{t^*}| \geq 3$ and t^* lies in the interior of the convex hull of V_{t^*}.*

Lemma 1 immediately implies the lower bound on $|\Pi(s^*, t^*)|$ when $s^* \in V$ or $t^* \in V$ since $|\Pi(s^*, t^*)| \geq \max\{|V_{s^*}|, |V_{t^*}|\}$. This finishes the proof for Cases **(V-*)**. Note that the bounds for Case **(V-V)** are trivial.

From now on, we assume that both s^* and t^* are not corners of $\mathcal{P}$. This assumption, together with Lemma 1, implies multiple shortest paths between s^* and t^*, and thus $\mathrm{d}(s^*, t^*) > \|s^* - t^*\|$. Hence, as discussed in Section 2, any maximal pair falling into one of Cases **(B-B)**, **(B-I)**, and **(I-I)** appears as a local maximum of the lower envelope of some path-length functions.

*Case **(I-I)**: When both s^* and t^* lie in* $\mathrm{int}\mathcal{P}$. We will apply Theorem 1 to prove Theorem 2 for Case **(I-I)**. By definition of u_i and v_i, we have that the path-length function $\mathrm{len}_{u_i,v_i}(s^*, t^*)$ satisfies $\mathrm{len}_{u_i,v_i}(s^*, t^*) = \mathrm{d}(s^*, t^*)$. Thus, at least m pairs (u, v) of corners satisfy $\mathrm{len}_{u,v}(s^*, t^*) = \mathrm{d}(s^*, t^*)$. If there are exactly m such pairs, we can apply Theorem 1 directly.

Unfortunately, this is not always the case. A single shortest path $\pi_i \in \Pi(s^*, t^*)$ may give additional pairs (u, v) of corners with $u, v \in \pi_i$ such that $(u, v) \neq (u_i, v_i)$ and $\mathrm{len}_{u,v}(s^*, t^*) = \mathrm{d}(s^*, t^*)$. This happens only when u, u_i, s^* or v, v_i, t^* are collinear. In order to resolve this problem, we define the *merged* path-length functions that satisfy all the requirements of Theorem 1 even in the degenerate case.

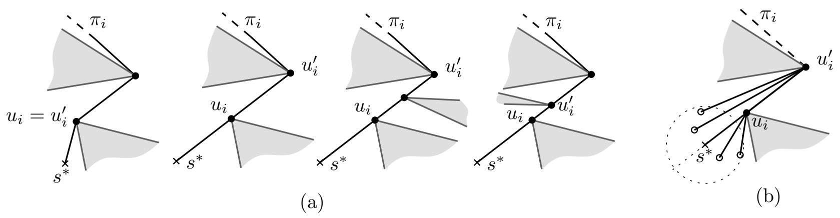


Fig. 2. (a) How to determine u'_i. (left to right) $u_i = u'_i$; s^*, u_i, and the second corner are collinear; s^* and the first three corners are collinear (b) For points in a small disk B centered at s^* with $B \subset \mathsf{VR}(u'_i) \cup \mathsf{VR}(u_i)$, the function start_i measures the length of the shortest path from u'_i to each.

Recall that the combinatorial structure of each shortest path $\pi_i \in \Pi(s^*, t^*)$ can be represented by a sequence $(u_i = u_{i,1}, \ldots, u_{i,k} = v_i)$ of corners in V (see Fig. 2). We define u'_i to be one of the $u_{i,j}$ as follows: If s^* does not lie on the line ℓ through u_i and $u_{i,2}$, then $u'_i := u_i$; otherwise, if $s^* \in \ell$, then $u'_i := u_{i,j}$, where j is the largest index such that for any open neighborhood $N \subset \mathbb{R}^2$ of s^* there exists a point $s \in (N \cap \mathsf{VR}(u_{i,j})) \setminus \ell$. Note that such u'_i always exists, and if no three of V are collinear, then we always have either $u'_i = u_i$ or $u'_i = u_{i,2}$; Fig. 2(a) illustrates how to determine u'_i. Also, we define v'_i in an analogous way. Let $\mathrm{start}_i\colon \mathsf{VR}(u'_i) \cup \mathsf{VR}(u_i) \to \mathbb{R}$ and $\mathrm{end}_i\colon \mathsf{VR}(v'_i) \cup \mathsf{VR}(v_i) \to \mathbb{R}$ be two functions defined as

$$\mathrm{start}_i(s) := \begin{cases} \|s - u'_i\| & \text{if } s \in \mathsf{VR}(u'_i), \\ \|s - u_i\| + \|u_i - u'_i\| & \text{if } s \in \mathsf{VR}(u_i) \setminus \mathsf{VR}(u'_i); \end{cases}$$

$$\mathrm{end}_i(t) := \begin{cases} \|t - v'_i\| & \text{if } t \in \mathsf{VR}(v'_i), \\ \|t - v_i\| + \|v_i - v'_i\| & \text{if } t \in \mathsf{VR}(v_i) \setminus \mathsf{VR}(v'_i). \end{cases}$$

This allows us to define merged path-length function $f_i\colon D_i \to \mathbb{R}$ as

$$f_i(s,t) := \mathrm{start}_i(s) + \mathrm{d}(u'_i, v'_i) + \mathrm{end}_i(t),$$

where $D_i := (\mathsf{VR}(u'_i) \cup \mathsf{VR}(u_i)) \times (\mathsf{VR}(v'_i) \cup \mathsf{VR}(v_i)) \subseteq \mathcal{P} \times \mathcal{P}$; see Fig. 2(b). We consider $\mathcal{P} \times \mathcal{P}$ as a subset of $\mathbb{R}^4$ and each pair $(s,t) \in \mathcal{P} \times \mathcal{P}$ as a point in $\mathbb{R}^4$. Also, we denote by (s_x, s_y) the coordinates of a point $s \in \mathcal{P}$ and we write $s = (s_x, s_y)$ or $(s,t) = (s_x, s_y, t_x, t_y)$ by an abuse of notation. Observe that $f_i(s,t) = \min\{\mathrm{len}_{u_i,v_i}(s,t),\ \mathrm{len}_{u'_i,v_i}(s,t), \mathrm{len}_{u_i,v'_i}(s,t), \mathrm{len}_{u'_i,v'_i}(s,t)\}$ for any $(s,t) \in D_i$ if we define $\mathrm{len}_{u,v}(s,t) = \infty$ when $s \notin \mathsf{VR}(u)$ or $t \notin \mathsf{VR}(v)$.

Lemma 2. *The functions f_i satisfy the following conditions.*

(i) There exists an open convex neighborhood $C \subset \mathbb{R}^4$ of (s^*, t^*) with $C \subseteq \bigcap D_i$ such that $\mathrm{d}(s,t) = \min_{i \in \{1,\ldots,m\}} f_i(s,t)$ for any $(s,t) \in C$.

(ii) All functions f_i are convex on C.

(iii) *None of the* f_i *attains a local minimum at* (s^*, t^*).
(iv) $f_i(s^*, t^*) = \mathrm{d}(s^*, t^*)$ *for any* $i \in \{1, \ldots, m\}$.
(v) *For any* $i \in \{1, \ldots, m\}$ *and any* $(s, t) \in \mathrm{int}D_i$*, there exists a unique line* $\ell_i \subset \mathbb{R}^4$ *through* (s, t) *such that* f_i *is constant on* $\ell_i \cap C$*. Moreover, there exists at most one other index* $j \neq i$ *such that* $\ell_i = \ell_j$.
(vi) *For any two indices* $i, j \in \{1, \ldots, m\}$*, any* $(s, t) \in C$ *and any small neighborhood* $U \subseteq C$ *of* (s, t)*, there exists a pair* $(s', t') \in U$ *such that* $f_i(s, t) < f_i(s', t')$ *and* $f_j(s, t) < f_j(s', t')$.

We take the convex neighborhood C of (s^*, t^*) of property (i) in Lemma 2 and apply Theorem 1 (note that properties (i)-(iv) ensure that the conditions of Theorem 1 are satisfied). Suppose that $m < 5$. Then, by Theorem 1, there exists at least one line $\ell \in \mathbb{R}^4$ through (s^*, t^*) such that d is constant on $\ell \cap C$. In the following, we will use the other geometric properties (v)-(vi) of Lemma 2 to achieve a contradiction, finally showing that $m \geq 5$ must hold.

Since (s^*, t^*) is a local maximum, there exists a small neighborhood $U \subset C$ of (s^*, t^*) such that $\mathrm{d}(s, t) \leq \mathrm{d}(s^*, t^*)$ for all $(s, t) \in U$. By property (v) of Lemma 2, at most two functions f_i are constant on $\ell \cap U$. Without loss of generality, we can assume that functions $f_3, \ldots, f_m$ are not constant. Since the geodesic distance is constant on $\ell \cap U$ and $\mathrm{d}(s, t) = \min_{i \in \{1, \ldots, m\}} f_i(s, t)$, any function that does not remain constant must strictly increase in both directions along ℓ.

That is, for any $(s', t') \in \ell \cap U$ with $(s', t') \neq (s^*, t^*)$ and for all $i \geq 3$, we have $\min\{f_1(s', t'), f_2(s', t')\} < f_i(s', t')$. Thus, there exists a small neighborhood $U_2 \subseteq U$ of (s', t') such that $\mathrm{d}(s, t) = \min\{f_1(s, t), f_2(s, t)\}$ for all $(s, t) \in U_2$. However, by property (vi) of Lemma 2, there exists a pair $(s'', t'') \in U_2$ such that $f_1(s', t') < f_1(s'', t'')$ and $f_2(s', t') < f_2(s'', t'')$, contradicting the fact that (s^*, t^*) is maximal. Hence, we achieve a bound $m = |\Pi(s^*, t^*)| \geq 5$, as claimed in Case **(I-I)** of Theorem 2.

As aforementioned, the other cases **(B-B)** and **(B-I)** can be handled with analogous arguments. The claimed bounds on $|V_{s^*}|$ and $|V_{t^*}|$ are shown by Lemma 1, which completes the proof of Theorem 2.

4 Computing the Geodesic Diameter

Since a diametral pair is in fact maximal, it falls into one of the cases shown in Theorem 2. In order to find a diametral pair we examine all possible scenarios accordingly.

Cases **(V-*)**, where at least one point is a corner in V, can be handled in $O(n^2 \log n)$ time by computing $\mathsf{SPM}(v)$ for every $v \in V$ and traversing it to find the farthest point from v, as discussed in Section 2. We focus on Cases **(B-B)**, **(B-I)**, and **(I-I)**, where a diametral pair consists of two non-corner points.

From the computational point of view, the most difficult case corresponds to Case **(I-I)** of Theorem 2. In particular, when $|\Pi(s^*, t^*)| = 5$ and ten corners of V are involved, resulting in $|V_{s^*}| = |V_{t^*}| = 5$. Note that we do not need to take special care for the case of $|\Pi(s^*, t^*)| > 5$. By Theorem 2 and its proof, it is guaranteed that there are at least five distinct pairs $(u_1, v_1), \ldots, (u_5, v_5)$ of corners in V such that $\text{len}_{u_i,v_i}(s^*, t^*) = \text{d}(s^*, t^*)$ for any $i \in \{1, \ldots, 5\}$ and the system of equations $\text{len}_{u_1,v_1}(s,t) = \cdots = \text{len}_{u_5,v_5}(s,t)$ determines a 0-dimensional zero set, corresponding to a constant number of candidate pairs in $\text{int}\mathcal{P} \times \text{int}\mathcal{P}$. Moreover, each path-length function $\text{len}_{u,v}$ is an algebraic function of degree at most 4. Given five distinct pairs (u_i, v_i) of corners, we can compute all candidate pairs (s,t) in $O(1)$ time by solving the system.[2] For each candidate pair we compute the geodesic distance between the pair to check its validity. Since the geodesic distance between any two points $s, t \in \mathcal{P}$ can be computed in $O(n \log n)$ time [13], we obtain a brute-force $O(n^{11} \log n)$-time algorithm, checking $O(n^{10})$ candidate pairs obtained from all possible combinations of 10 corners in V.

As a different approach, one can exploit the SPM-*equivalence decomposition* of $\mathcal{P}$, which subdivides $\mathcal{P}$ into regions such that the shortest path map of any two points in a common region are *topologically equivalent* [8]. It is not difficult to see that if (s,t) is a pair of points that equalizes any five path-length functions, then both s and t appear as vertices of the decomposition. However, the current best upper bound on the complexity of the SPM-equivalence decomposition is $O(n^{10})$ [8], and thus this approach hardly leads to a remarkable improvement.

Instead, we do the following for Case **(I-I)** with $|V_{s^*}| = 5$. We choose any five corners $u_1, \ldots, u_5 \in V$ (as a candidate for the set V_{s^*}) and overlay their shortest path maps $\mathsf{SPM}(u_i)$. Since each $\mathsf{SPM}(u_i)$ has $O(n)$ complexity, the overlay consists of $O(n^2)$ cells. Any cell of the overlay is the intersection of five cells associated with $v_1, \ldots, v_5 \in V$ in $\mathsf{SPM}(u_1), \ldots, \mathsf{SPM}(u_5)$, respectively. Choosing a cell of the overlay, we get five (possibly, not distinct) $v_1, \ldots, v_5$ and a constant number of candidate pairs by solving the system $\text{len}_{u_1,v_1}(s,t) = \cdots = \text{len}_{u_5,v_5}(s,t)$. We iterate this process for all possible tuples of five corners $u_1, \ldots, u_5$, obtaining a total of $O(n^7)$ candidate pairs in $O(n^7 \log n)$ time. Note that the other subcases with $|V_{s^*}| \leq 4$ can be handled similarly, resulting in $O(n^6)$ candidate pairs.

For each candidate pair (s,t), we must check their validity (that is, that the paths $(s, u_i, \ldots, v_i, t)$ are indeed shortest paths) and their geodesic distance using a two-point query structure of Chiang and Mitchell [8]. For a fixed parameter $0 < \delta \leq 1$ and any fixed $\epsilon > 0$, we can construct, in $O(n^{5+10\delta+\epsilon})$ time, a data structure that supports $O(n^{1-\delta} \log n)$-time two-point shortest path queries. The total running time is $O(n^7 \log n) + O(n^{5+10\delta+\epsilon}) + O(n^7) \times O(n^{1-\delta} \log n)$. We set $\delta = \frac{3}{11}$ to optimize the running time to $O(n^{7+\frac{8}{11}+\epsilon})$.

Also, we can use an alternative two-point query data structure whose performance is sensitive to h [8]: after $O(n^5)$ preprocessing time using $O(n^5)$ storage, two-point queries can be answered in $O(\log n + h)$ time.[3] Using this alternative

[2] Here, we assume that fundamental operations on a constant number of polynomials of constant degree with a constant number of variables can be performed in constant time.

[3] If h is relatively small, one could use the structure of Guo, Maheshwari and Sack [11] which answers a two-point query in $O(h \log n)$ time after $O(n^2 \log n)$ preprocessing time using $O(n^2)$ storage, or another structure by Chiang and Mitchell [8] that supports a two-point query in $O(h \log n)$ time, spending $O(n + h^5)$ preprocessing time and storage.

structure, the total running time of our algorithm becomes $O(n^7(\log n + h))$. Note that this method outperforms the previous one when $h = O(n^{\frac{8}{11}})$.

The other cases can be handled analogously with strictly better time bound. For Case **(B-I)**, we handle only the case of $|\Pi(s^*, t^*)| = 4$ with $|V_{t^*}| = 3$ or 4. For the subcase with $|V_{t^*}| = 4$, we choose any four corners from V as $v_1, \ldots, v_4$ as a candidate for V_{t^*} and overlay their shortest path maps $\mathsf{SPM}(v_i)$. The overlay, together with V, decomposes $\partial\mathcal{P}$ into $O(n)$ intervals. Each such interval determines $u_1, \ldots, u_4$ as above, and the side $e_s \in E$ on which s^* should lie. Now, we have a system of four equations on four variables: three from the corresponding path-length functions len_{u_i,v_i} which should be equalized at (s^*, t^*) and the fourth from the supporting line of e_s. Solving the system, we get a constant number of candidate maximal pairs, again by Theorem 2 and its proof. In total, we obtain $O(n^5)$ candidate pairs. The other subcase with $|V_{t^*}| = 3$ can be handled similarly, resulting in $O(n^4)$ candidate pairs. As above, we can exploit two different structures for two-point queries. Consequently, we can handle Case **(B-I)** in $O(n^{5+\frac{10}{11}+\epsilon})$ or $O(n^5(\log n + h))$ time.

In Case **(B-B)** when $s^*, t^* \in \mathcal{B}$, we handle the case of $|\Pi(s^*, t^*)| = 3$ with $|V_{s^*}| = 2$ or 3. For the subcase with $|V_{s^*}| = 3$, we choose three corners as a candidate of V_{s^*} and take the overlay of their shortest path maps $\mathsf{SPM}(u_i)$. It decomposes $\partial\mathcal{P}$ into $O(n)$ intervals. Each such interval determines three corners v_1, v_2, v_3 forming V_{t^*} and a side $e_t \in E$ on which t^* should lie. Note that we have only three equations so far; two from the three path-length functions and the third from the line supporting to e_t. Since s^* also should lie on a side $e_s \in E$ with $e_s \neq e_t$, we need to fix such a side e_s that $\bigcap_{1\le i\le 3} \mathsf{VR}(u_i)$ intersects e_s. In the worst case, the number of such sides e_s is $\Theta(n)$. Thus, we have $O(n^5)$ candidate pairs for Case **(B-B)**; again, the other subcase with $|V_{s^*}| = 2$ contributes to a smaller number $O(n^4)$ of candidate pairs. Testing each candidate pair can be done as above, resulting in $O(n^{5+\frac{10}{11}+\epsilon})$ or $O(n^5(\log n + h))$ total running time.

Alternatively, one can exploit a two-point query structure only for boundary points on $\partial\mathcal{P}$ for Case **(B-B)**. The two-point query structure by Bae and Okamato [6] builds an explicit representation of the graph of the lower envelope of the path-length functions $\mathrm{len}_{u,v}$ restricted on $\partial\mathcal{P} \times \partial\mathcal{P}$ in $O(n^5 \log n \log^* n)$ time.[4] Since $|\Pi(s^*, t^*)| \geq 3$ in Case **(B-B)**, such a pair appears as a vertex on the lower envelope. Hence, we are done by traversing all the vertices of the lower envelope.

As Case **(I-I)** is the bottleneck, we conclude the following.

Theorem 3. *Given a polygonal domain having n corners and h holes, the geodesic diameter and a diametral pair can be computed in $O(n^{7+\frac{8}{11}+\epsilon})$ or $O(n^7(\log n + h))$ time in the worst case, where ϵ is any fixed positive number.* ⊔⊓

5 Concluding Remarks

We have presented the first algorithms that compute the geodesic diameter of a given polygonal domain. As mentioned in the introduction, a similar result for

[4] More precisely, in $O(n^4\lambda_{65}(n)\log n)$ time, where $\lambda_m(n)$ stands for the maximum length of a Davenport-Schinzel sequence of order m on n symbols.

convex 3-polytopes was shown in [16]. We note that, although the main result of this paper is similar, the techniques used in the proof are quite different. Indeed, the key requirement for our proof is the fact that shortest paths in our environment are polygonal chains whose vertices are in V, a claim that does not hold in higher dimensions (even in 2.5-D surfaces). It would be interesting to find other environments in which similar result holds.

Another interesting question would be finding out how many maximal pairs a polygonal domain can have. The analysis of Section 4 gives an $O(n^7)$ upper bound. On the other hand, one can easily construct a simple polygon in which the number of maximal pairs is $\Omega(n^2)$. Any improvement on the $O(n^7)$ upper bound would lead to an improvement in the running time of our algorithm.

Though in this paper we have focused on *exact* geodesic diameters only, an efficient algorithm for finding an *approximate* geodesic diameter would be also interesting. Notice that any point $s \in \mathcal{P}$ and its farthest point $t \in \mathcal{P}$ yield a 2-approximate diameter; that is, $\mathrm{diam}(\mathcal{P}) \leq 2\max_{t\in\mathcal{P}} \mathrm{d}(s,t)$ for any $s \in \mathcal{P}$. Also, based on a standard technique using a rectangular grid with a specified parameter $0 < \epsilon < 1$, one can obtain a $(1+\epsilon)$-approximate diameter in $O((\frac{n}{\epsilon^2} + \frac{n^2}{\epsilon})\log n)$ time as follows. Scale $\mathcal{P}$ so that $\mathcal{P}$ can fit into a unit square, and partition $\mathcal{P}$ with a grid of size $\epsilon^{-1} \times \epsilon^{-1}$. We define the set D as the point set that has the center of grid squares (that have a nonempty intersection with $\mathcal{P}$) and intersection points between boundary edges and grid segments. We now can discretize the diameter problem by considering only geodesic distances between pairs of points of D. It turns out that, when ϵ is small enough, the distance between any two points s and t in $\mathcal{P}$ is within a $1+\epsilon$ factor of the distance between two points of D. Breaking the quadratic bound in n for the $(1+\epsilon)$-approximate diameter seems a challenge at this stage.[5] We conclude by posing the following problem: for any or some $0 < \epsilon < 1$, is there any algorithm that finds a $(1+\epsilon)$-approximate diametral pair in $O(n^{2-\delta} \cdot \mathrm{poly}(1/\epsilon))$ time for some positive $\delta > 0$?

Acknowledgements. We thank Hee-Kap Ahn, Jiongxin Jin, Christian Knauer, and Joseph Mitchell for fruitful discussion, and Joseph O'Rourke for pointing out the reference [20].

References

1. Agarwal, P.K., Aronov, B., O'Rourke, J., Schevon, C.A.: Star unfolding of a polytope with applications. SIAM J. Comput. 26(6), 1689–1713 (1997)
2. Aronov, B., Fortune, S., Wilfong, G.: The furthest-site geodesic Voronoi diagram. Discrete Comput. Geom. 9, 217–255 (1993)
3. Asano, T., Toussaint, G.: Computing the geodesic center of a simple polygon. Technical Report SOCS-85.32, McGill University (1985)
4. Bae, S.W., Chwa, K.-Y.: The geodesic farthest-site Voronoi diagram in a polygonal domain with holes. In: Proc. 25th Annu. Sympos. Comput. Geom. (SoCG), pp. 198–207 (2009)

[5] The idea of this approximation algorithm is due to Hee-Kap Ahn.

5. Bae, S.W., Korman, M., Okamoto, Y.: The geodesic diameter of polygonal domains. arXiv preprint (2010), arXiv:1001.0695
6. Bae, S.W., Okamoto, Y.: Querying two boundary points for shortest paths in a polygonal domain. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 1054–1063. Springer, Heidelberg (2009), arXiv:0911.5017
7. Chazelle, B.: A theorem on polygon cutting with applications. In: Proc. 23rd Annu. Sympos. Found. Comput. Sci. (FOCS), pp. 339–349 (1982)
8. Chiang, Y.-J., Mitchell, J.S.B.: Two-point Euclidean shortest path queries in the plane. In: Proc. 10th ACM-SIAM Sympos. Discrete Algorithms (SODA), pp. 215–224 (1999)
9. Cook IV, A.F., Wenk, C.: Shortest path problems on a polyhedral surface. In: Proc. 11th Internat. Sympos. Algo. Data Struct (WADS), pp. 156–167 (2009)
10. Guibas, L.J., Hershberger, J.: Optimal shortest path queries in a simple polygon. J. Comput. Syst. Sci. 39(2), 126–152 (1989)
11. Guo, H., Maheshwari, A., Sack, J.-R.: Shortest path queries in polygonal domains. In: Fleischer, R., Xu, J. (eds.) AAIM 2008. LNCS, vol. 5034, pp. 200–211. Springer, Heidelberg (2008)
12. Hershberger, J., Suri, S.: Matrix searching with the shortest path metric. SIAM J. Comput. 26(6), 1612–1634 (1997)
13. Hershberger, J., Suri, S.: An optimal algorithm for Euclidean shortest paths in the plane. SIAM J. Comput. 28(6), 2215–2256 (1999)
14. Mitchell, J.S.B.: Shortest paths among obstacles in the plane. Internat. J. Comput. Geom. Appl. 6(3), 309–331 (1996)
15. Mitchell, J.S.B.: Shortest paths and networks. In: Handbook of Discrete and Computational Geometry, 2nd edn., ch. 27, pp. 607–641. CRC Press, Inc., Boca Raton (2004)
16. O'Rourke, J., Schevon, C.: Computing the geodesic diameter of a 3-polytope. In: Proc. 5th Annu. Sympos. Comput. Geom. (SoCG), pp. 370–379 (1989)
17. O'Rourke, J., Suri, S.: Polygons. In: Handbook of Discrete and Computational Geometry, 2nd edn., ch. 26, pp. 583–606. CRC Press, Inc., Boca Raton (2004)
18. Pollack, R., Sharir, M., Rote, G.: Computing the geodesic center of a simple polygon. Discrete Comput. Geom. 4(6), 611–626 (1989)
19. Suri, S.: The all-geodesic-furthest neighbors problem for simple polygons. In: Proc. 3rd Annu. Sympos. Comput. Geom. (SoCG), pp. 64–75 (1987)
20. Zalgaller, V.A.: An isoperimetric problem for tetrahedra. Journal of Mathematical Sciences 140(4), 511–527 (2007)

Polyhedral and Algorithmic Properties of Quantified Linear Programs[*]

Ulf Lorenz, Alexander Martin, and Jan Wolf

Institute of Mathematics, Technische Universität Darmstadt, Germany

Abstract. Quantified linear programs (QLPs) are linear programs with variables being either existentially or universally quantified. The integer variant is PSPACE-complete, and the problem is similar to games like chess, where an existential and a universal player have to play a two-person-zero-sum game. At the same time, a QLP with n variables is a variant of a linear program living in $\mathbb{R}^n$, and it has strong similarities with multistage-stochastic programs with variable right-hand side. We show for the continuous case that the union of all winning policies of the existential player forms a polytope in $\mathbb{R}^n$, that its vertices are games of so called extremal strategies, and that these vertices can be encoded with polynomially many bits. The latter allows the conclusion that solving a QLP is in PSPACE. The hardness of the problem stays unknown.

1 Introduction

In the 1940s, linear programming arose as a mathematical planning model and rapidly found its daily use in many industries. However, integer programming, which was introduced in 1951, became dominant far later at the beginning the 1990s. Certainly, one reason for the delay of the integer programming success story stems from the fact that linear programming resides in the complexity class P, while integer programming is NP complete. Nowadays, we are able to solve very large mixed integer programs of practical size, but companies observe an increasing danger of disruptions, i.e., events occur which prevent companies from acting as planned. Therefore, there is a need for planning and deciding under uncertainty. Uncertainty, however, often pushes the complexity of traditional optimizations problems, which are in P or NP, to PSPACE. The quantified versions of linear integer programs cover the complexity class PSPACE. The relaxed versions, which we examine in this paper, additionally have remarkable polyhedral properties. The idea of our research is to explore the abilities of linear programming when applied to PSPACE-complete problems, similar as it was applied to NP-complete problems in the 1990s.

1.1 State-of-the-Art

For traditional deterministic optimization one assumes data for a given problem to be fixed and exactly known when the decisions have to be taken. However, data

[*] Research partially supported by German Research Foundation (DFG) funded SFB 805.

M. de Berg and U. Meyer (Eds.): ESA 2010, Part I, LNCS 6346, pp. 512–523, 2010.

are often afflicted with some kinds of uncertainties, and only estimations, maybe in form of probability distributions, are known. Examples are flight or travel times. Throughput-time, arrival times of externally produced goods, and scrap rate are subject to variations in production planning processes. One possibility to deal with these uncertainties is to aggregate a given probability distribution to a single estimated number. Then, the optimum concerning these estimated input data can be computed with the help of traditional optimization tools. In some fields of application, as e.g. the fleet assignment problem of airlines, this procedure was successfully established. In other fields, like production planning and control, this technique could not be successfully applied, although mathematical models do exist [17]. Because of the intuitive complexity of even some deterministic models, its stochastic counterparts are often not considered. Instead, safety stock is introduced, demanded quantities are overestimated, and buffers are oversized. However, this resource intensive behaviour is contrasted by a couple of publications within the last decade, indicating that stochastic problems are not necessarily out of scope [7,12,15,14,19].

1.2 Complexity and Algorithmic Issues

From complexity theory, we know that many interesting optimization problems under uncertainty are PSPACE complete [16]. The negative results from complexity theory seem discouraging to find efficient (i.e. polynomial time) algorithms to solve optimization problems that are PSPACE complete. As a consequence, algorithmics mostly deal with restricted problems which allow finding a polynomial-time algorithm, or at least approximations [10,6]. The self-restriction to *efficient algorithms* (in a formal sense), however, is related to worst-case instances. This does not reflect the fact that the most astonishing and admired successes of computing intelligence are modeled as NP-complete problems (mixed integer linear programs) and PSPACE-complete problems (computer games like Chess [4]).

Contrasting the efficiency debate, we can interpret a set of expressions, which can encode a PSPACE-complete problem, as a very powerful modeling language: more powerful than necessary to encode any NP-complete problem. The fact that it is not possible to find polynomial time algorithms for all problems that are encoded with the help of such a powerful modeling language, leads to the consequence that research for new solutions must be driven from the application-side or even from the instances-side, as e.g. presented in [12]. Relatively unexplored are the abilities of linear programming extensions in the PSPACE-complete world. In this context, Subramani introduced the notion of quantified linear programs [20,21].

1.3 Solution Issues

Prominent solution paradigms for optimization under uncertainty are Dynamic Programming [2], Sampling [11], the exploration of Markov-Chains [22], Robust Optimization [13], and Stochastic Programming [3,7,18,5]. Markov-Chains and

Dynamic Programming are often used for problems from complexity class P, but for more complex problems, other algorithms are often faster, like e.g. the Alphabeta algorithm for two-person zero-sum games [4]. Stochastic Programming can be essentially divided into two-stage and multi-stage problems with so-called recourse. A set of initial decisions are taken first, followed by a random event. After this, recourse decisions are taken, which allow to compensate for events that have been observed in previous stages. The multi-stage problem, accordingly consists of multiple stages, with a random event occurring between each stage. Such a problem can be transformed into a so-called deterministic equivalent program (DEP) and then be solved with the help of linear programming. An appropriate procedure for solving multi-stage stochastic programs are the nested decomposition procedure and its variants [3]. A multi-stage stochastic integer problem with only discrete probabilities is also called a game against nature, cf. [16,8].

In Section 2, we formally describe the QLP-problem and present an illustrating example. In Section 3, we analyse the polyhedral properties of the QLP problem. We make intensive use of the problem's ambiguity, being a two-person zero-sum game between two players on the one side, and being a convex multi-stage decision problem on the other one. In the last section, we present a solution algorithm running in polynomial space, based on Nested Benders decomposition. Up to now, only Fourier-Motzkin elimination based procedures were known, which consume double exponential time and space in the worst case.

2 The Problem Statement: Quantified Linear Programs

Within this paper, we intend to concentrate on quantified linear programs, as they were introduced by Subramani [21,20], who also presented first analytical results.

Given: A vector of variables $x = (x_1, ..., x_n) \in \mathbb{Q}^n$, upper and lower bounds $u \in \mathbb{Z}^n$ and $l \in \mathbb{Z}^n$ with $l_i \leq x_i \leq u_i$, a matrix $A \in \mathbb{Q}^{m\times n}$, a vector $b \in \mathbb{Q}^m$ and a quantifier string $Q = (q_1, ..., q_n) \in \{\forall, \exists\}^n$, where the quantifier q_i belongs to the variable x_i, for all $1 \leq i \leq n$. We denote a QLP as $[Q : Ax \leq b]$. A maximal subset of $Q(x)$, which contains a consecutive sequence of quantifiers of the same type, is called a (variable-) *block*. A full assignment of numbers to the variables is interpreted as a game between an existential player (fixing the existential variables) and a universal player (fixing the universally quantified variables). The variables are set in consecutive order, as determined by the quantifier string. Consequently, we say that a player makes the move $x_k = z$, if he fixes the variable x_k to the value z. At each such move, the corresponding player knows the settings of $x_1, ..., x_{i-1}$ before setting x_i. If, at the end, all constraints of $Ax \leq b$ hold, the existential player wins the game. Otherwise the universal player wins. For ease of explanation, we sometimes rewrite the system as $[Q(x, y) : A \cdot \binom{x}{y} \leq b]$, y representing the universal variables, x the existential ones. Without loss of generality, we can assume that universally quantified variables are between zero and one.

$$\forall x_1 \forall x_2 \exists x_3 : \begin{pmatrix} 10 & -4 & 2 \\ 10 & 4 & -2 \\ -10 & 4 & 1 \\ -10 & -4 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 4 \\ 12 \\ 8 \end{pmatrix}, \textit{with} \quad x_1 \in [-1,0], x_2 \in [0,1], x_3 \in [-2,2]$$

Fig. 1. QLP instance in dimension 3

Question: Is there an algorithm that fixes a variable x_i with the knowledge, how $x_1, ..., x_{i-1}$ have been set before, such that the existential player wins the game, no matter, how the universal player acts when he has to move?

In [20,21], this problem occurs in two variants: a) all variables are discrete (QIP) and b) all variables are continuous (QLP). It has been shown that the restricted QLP problem with only one quantifier-change is either in P (when the quantifier string begins with existential quantifiers and ends with universal ones) or is coNP-complete (when the quantifier string begins with universal quantifiers and ends with existential ones) [21].

Definition 1. *(Strategy) A* strategy $(V_x \dot{\cup} V_y, E, L)$ *for the existential player S is a labeled tree of depth n, withV_x and V_y being two disjoint sets of nodes, and a set $L \in \mathbb{Q}^{|E|}$ of edge-labels. Nodes from V_x are called* existential nodes*, nodes from V_y are called* universal nodes*. Each tree level i consists either of only existential nodes or of only universal nodes, depending on the quantifier q_i of the variable x_i. Each edge of the set E, leaving a tree-node of level i, represents an assignment for the variable x_i. $l_i \in L$ describes the value of variable x_i on edge $e_i \in E$. Existential nodes have exactly one successor, universal nodes have two successors, one representing $x_i = 0$ and $x_i = 1$ the other*[1]*. (Generally, the coefficients of a universal variable y of an arbitrary QLP-instance can be scaled in a way such that $y \in [0,1]$.) A strategy is called a* winning strategy*, if all paths from the root to a leaf represent a vector x such that $Ax \leq b$.*

Definition 2. *(Policy) A* policy *is an algorithm that fixes a variable x_i, being the i^{th} component of the vector x, with the knowledge, how $x_1, ..., x_{i-1}$ have been set before.*

A 3-dimensional example of a QIP/QLP is shown in Fig. 1. If we restrict the variables to the integer bounds of their domains, we observe a winning strategy for the existential player as shown in Figure 2. In this example, a + in a tree leaf means that the existential player wins when this leaf is reached. A - marks a win for the universal player. Numbers at the edges mark the choices for variables. If the universal player moves to -1 and 0 (i.e., he sets $x_1 = -1$ and $x_2 = 0$) the existential player has to move to 2. If the universal player moves to -1 and 1, the existential player must set the variable $x_3 = -2$ etc. We see that the existential player has to react carefully to former moves of the universal player.

[1] Later, in Lemma 2, we will show that by forcing universal variables to 0 and 1, we do not loose any generality.

If we now relax the variables, allowing non-integral values for the corresponding domains, the resulting solution space of the corresponding QLP becomes polyhedral.

Moreover, the solution of the resulting problem when $x_1 = -1$ is a line segment: B (cf. Fig 3). On the left side of Figure 3, we can still see the corresponding partial strategy of the integer example in Figure 2. The strategy consists of the two end-points of B. On the right side of Figure 3, the same convex hull of the solution space is shown from another perspective.

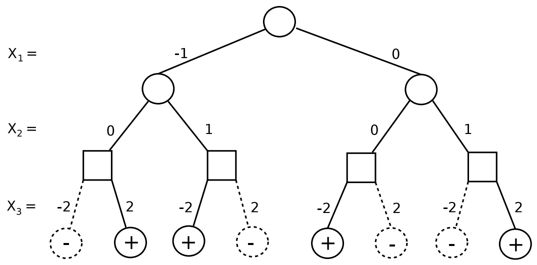


Fig. 2. The winning strategy (solid) for the integer QLP example in Fig. 1

We observe that the existential player has more freedom in the choice of x_3, when the universal player sets $x_1 = 0$. If $x_1 = 0$, the solution space of the rest-problem will just be the facet C. Interestingly, the line segment B and the facet C lie in the 3-dimensional vector space in such a way that there exists no linear description for a solution policy. The fact that a solution of a QLP has no linear description in general is not surprising, because the QLP problem is coNP-hard.

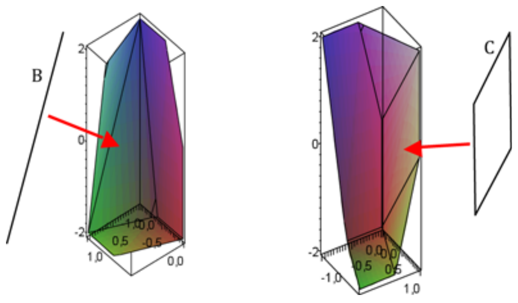


Fig. 3. A visualization of the 3-d solution space of example of Fig. 1

3 QLP Analysis

In the following, we consider a QLP as defined in the previous section. Let existentially quantified variables be from $\mathbb{Q}$ and universally quantified variables be from $\{0, 1\}$, and let n be the dimension of x.

Definition 3. *(convex combination of strategies) Let S and S' be two strategies for the same QLP instance. We call $S'' = (V, E, L'')$ a convex combination of $S = (V, E, L)$ and $S' = (V, E, L')$ if there is some $\alpha \in [0, 1]$ such that for each edge $e_i \in E$ it is $l''_i = \alpha l'_i + (1 - \alpha)l_i$. We write $S'' = \alpha S' + (1 - \alpha)S$.*

Lemma 1. *(strategy-convexity) If S and S' are winning strategies (for the existential player), and $\alpha \in [0, 1]$, also $S'' = \alpha S' + (1 - \alpha)S$ will be a winning strategy.*

Proof. *(sketched) We apply conventional convexity to all paths of the strategies S, S' and S''.*

Up to this point, we have restricted the choices of the universal player to 0 and 1 because we were interested in the existence of winning policies for the existential player. Now, we are interested in polyhedral properties. Therefore, the next step is to relax this integrality constraint. As a consequence, the solution of an existential strategy can no longer be described as a tree, and instead, we have to use the notion of a policy. This means, an existential variable, i.e. the component x_k of a solution vector x is determined as the value of a computable function: $x_k = f_k(x_1, ..., x_{k-1})$.

Lemma 2. *(policy-convexity) If P and P' are winning policies for the existential player, also $P'' = \alpha P' + (1-\alpha)P$ will be a winning policy for the existential player. Here, P'' is an algorithm that picks the outputs of P and P' for arbitrary input and combines them accordingly.*

Proof. *(sketched) Conventional convexity arguments on paths of the policies P, P' and P'' are combined with induction.*

Remark: We can conclude that *the existential player has a winning strategy against a two-value-restricted universal player if and only if he has a winning policy against an universal player without the restriction (also cf. [21]).*

3.1 A Simple Solution Procedure for the QLP Problem ([21])

Let $[Q : Ax \leq b]$ be a QLP instance. If the last variable x_n of x is existentially quantified, we can apply the Fourier-Motzkin elimination (FME) procedure. No matter how $x_1, ..., x_{n-1}$ are fixed, the remaining problem is a (parametrized) polytope. If, however, x_n is a universal quantified variable, there is a simpler algorithm to eliminate the last variable. Let us interpret the given instance as a game between the two players. Then fixing the last variable means that the universal player makes the last move. The existential player wants to achieve that $Ax \leq b$, but the universal player wants to hinder him, especially with his last move. If possible, the universal player will therefore choose a move which destroys at least one of the inequalities. Therefore, the existential player has a winning strategy if and only if he has a winning strategy against the $(n-1)$-dimensional problem which occurs if we assume the worst case universal choice, inequality by inequality. Let us call the procedure that eliminates existential variables with the help of the FME and universal variables with the help of the given gaming argument, the *QLP-elimination-procedure.* A detailed step by step proof for correctness can be found in [21].

Definition 4. *(extremal strategies (extremal policy)) A strategy (policy) S is called* extremal*, if S cannot be described as a convex combination of two different strategies (policies). This includes that all sub-strategies / sub-policies, induced by the corresponding subtrees of S, are extremal, if and only if S is extremal itself.*

3.2 The Solution Spaces of QLP Instances

In the following, we show that the solution space of a QLP instance, i.e., the union of all winning policies of the existential player, is a polytope. Certainly, there are several ways how to show the given fact, but we choose a quite elementary approach, because it makes the following detail visible: The vertices of the solution-space correspond to games of extremal winning strategies. Moreover, an extremal winning strategy is determined by a subset of polytope-vertices in $\mathbb{R}^n$. The proof proceeds in several steps.

Lemma 3. *The convex combination of all leaves of all extremal winning strategies against a $\{0,1\}$-restricted universal player, is a polytope in $\mathbb{R}^n$.*

Proof. : Of course, the convex combination of the leaves of all winning strategies is a convex set. We now argue that the number of extreme points is finite. Let any QLP-instance $I = [Q : Ax \leq b]$ be given, let S be an extremal winning strategy for the existential player. We eliminate the variables $x_n, ..., x_{n-k}$ such that only the first block of variables stays, with the help of the QLP-elimination-procedure. If the variables of this first block are m universally quantified variables, the first move of the universal player is an element of $\{0,1\}^m$. If the variables of the first block are existential variables, the remaining problem is a conventional linear program with m variables. This remaining problem has finitely many extreme points, and S can only be extremal, if the first move of S corresponds to one of these extreme points, because otherwise S could be composed as a convex combination of other strategies. Therefore, there are only finitely many possible choices for the first player in any extremal strategy. Inductively, we see that there are only finitely many extremal strategies, and therefore there are only finitely many leaves within these strategies, and therefore the convex combination of all leaves of all extremal strategies contains only finitely many extreme points.

Lemma 4. *The convex combination of all outcomes of all possible winning policies is a polytope.*

Proof. Let P be a winning policy. If the universal player chooses moves only from $\{0,1\}$, the winning policy is a winning strategy. This strategy is either an extremal strategy, or each game of the strategy can be expressed as the convex combination of two other strategies. Thus, every point, which is represented by an existential-player-winning game, is also a point in the convex hull of the leaves of all extremal winning strategies. If the universal player is allowed to choose his moves from $\{0,\alpha,1\}$, the resulting game can be expressed as a convex combination of extremal strategies as well, for all $\alpha \in [0,1]$. Again, every point, which is represented by such a game, where the existential player wins, is also a point in the convex hull of the leaves of all extremal winning strategies. Last but not least, it is obvious that there are no holes in the union of the winning policies. Let p be any point in the convex hull of the extremal winning strategies, and let us interpret this point as a game between the existential player and the universal player. Then, this game is part of a winning policy, because otherwise the point could not be part of the convex hull of the extremal winning strategies.

Theorem 1. *The solution space of all winning policies is a polytope (clear with Lemma 3 and Lemma 4)*

Theorem 2. *The vertices of a QLP-solution space can be described with polynomial many bits per vertex.* In order to prove this theorem, we present some preparations. Especially, we make use of a technique known from stochastic programming, where a deterministic equivalent program (DEP) is built for a multistage stochastic program. The general form of a multistage stochastic program with fixed recourse is as follows (cf. [3]), where the matrices T^i and the vectors h^i are possibly stochastic.

$$\begin{array}{ll} \min_{x^0} & c^0x^0 + E_{\xi^1}\left(\min_{x^1} c^1x^1 + \ldots + E_{\xi^H|\xi^1\ldots\xi^{H-1}}\left(\min_{x^H} c^Hx^H\right)\right) \\ s.t. & Ax^0 = b, \\ & \mathbf{T^0}x^0 + Wx^1 = h^1 \\ & \ldots \\ & \mathbf{T^{H-1}}x^{H-1} + Wx^H = \mathbf{h}^H, \\ & l^t \leq x^t \leq u^t, \text{for all } x \in \{0,\ldots,H\} \end{array}$$

Usually, it is assumed that the stochastic elements are defined over a discrete probability space $(\Xi, \sigma(\Xi), P)$, where $\Xi = \Xi^1 \otimes \cdots \otimes \Xi^H$ is the support of the random data in each period, subject to $\Xi^t = \{\xi^t_s = (T^t_s, W^t_s, h^t_s, c^t_s), s = 1,\ldots,S^t\}$. Because the probability space is discrete, a DEP can be defined by replicating the deterministic linear program for each possible scenario (possible path of events). Additionally, it is required that decisions do not depend on future events. We can construct a deterministic equivalent for QLP-instances, accordingly. Instead to encode the scenario tree, we encode the decision tree of the universal player into the deterministic equivalent by replicating the inequalities of the QLP-instance for each possible decision combination of the universal player. Let $A_1, .., A_t$ be those blocks of columns of A that belong to the existential player and which are separated by the columns belonging to the universal player. If we bring the universal-player-columns to the right side, we will have created a multistage decision problem with variable right-hand side. Now, let $[Q : Ax \leq b]$ be a QLP instance with t quantifier-changes, and, furthermore, let $[Q' : Ax \leq b]$ be the QLP-instance where the sequence of quantifiers is modified to $(\exists \ldots \exists \forall \ldots \forall)$.

Figure 4a) shows the block structure of an example with four scenarios in block ladder matrix form. In the example, the universal player is allowed to make two moves, and there are two universally quantified variables and the quantifier-sequence is $\exists\forall\exists\forall\exists$. Figure 4c) shows the block structure of the manipulated QLP Q'. It can be observed that the dimension of the deterministic equivalent in Fig. 4 a) is higher, because the different choices of the existential player after a move of the universal player have to be considered. Figure 4a) and b) show, how the block structure of an instance changes, when the position of the last universal quantifier is changed.

Definition 5. *Let $P \subset \mathbb{R}^n$ be a polytope and ϕ and ν be positive integers.*

(i) P has row-complexity *(also called facet-complexity in [9]) of at most ϕ, if there is a system of inequalities with rational coefficients $A \in \mathbb{Q}^{m\times n}, b \in \mathbb{Q}^m$*

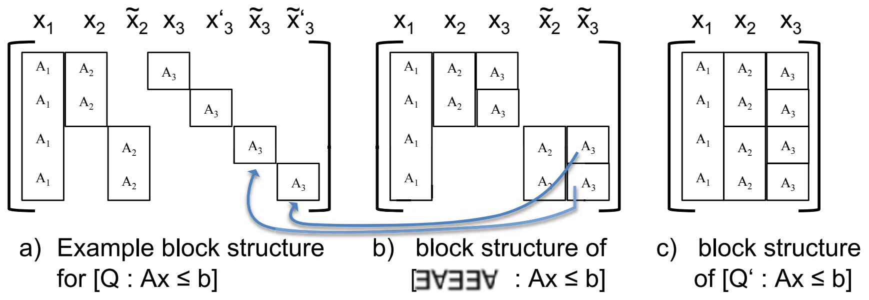

a) Example block structure for [Q : Ax ≤ b]
b) block structure of [∀∃∃∀∃ : Ax ≤ b]
c) block structure of [Q‘ : Ax ≤ b]

Fig. 4. Block structures of Q, Q’, and a medium instance

for some $m \in I\!N$, such that P is described with the help of a set of inequalities and the encoding length (i.e. the number of bits used) of each row is at most ϕ. Each entry in A and b requires at least one bit.

(ii) P has vertex-complexity *of at most ν, if there are finite sets $V, E \subseteq \mathbb{R}^n$, such that $P = convexhull(V)$ and the encoding length of each vector in $V \cup E$ is at most ν. We assume that $\nu \geq n$.*

Lemma 5. *$\nu \leq 4\phi n^2$, and $\phi \leq 3\nu n^2$.* Proof and details can be found in [9].

Lemma 6. *The vertex-complexity ν' of the manipulated QLP-polytope $P([Q' : Ax \leq b]), A \in Q^{m\times n}$, $b \in Q^m$ is in $O(\phi n^6)$, ϕ being the worst-case row-complexity of (A, b).*

Proof. Let the row-complexity of $QLP' := [Q' : Ax + By \leq b]$ be $\phi_{QLP'}$ with existential variables x and universal variables y, let the number of universal variables equal to k. QLP' has 2^k scenarios. Its DEP, denoted by $detEq(QLP')$, or $detEq'$ for short, lives in $\mathbb{R}^{n-k}$ and has row-complexity $\phi_{detEq'} \leq \phi_{QLP'}$ because the DEP simply consists of a replication of A with various right hand sides $b_i, i \in \{1, .., 2^k\}$. Moreover, the encoding length $\langle b_i \rangle$ is less or equal to $\langle By \rangle$ for any y. Let D be the set of inequalities of $detEq'$. (q.e.d.)

As a consequence of Lemma 5, the vertex-complexity $\nu_D \leq 4\phi_D n^2$. Now we analyse the application of the following algorithm, which creates a game of an extremal winning strategy for the existential player and thus a vertex of an implicit QLP-polytope.

```
Algorithm: compute some QLP-vertex ($\mathcal{Q}$)
1 $D^{\mathcal{Q}}$ := set of inequalities of deterministic equivalent of QLP-instance Q
2 for i := 1 to t-1 // let $t$ different blocks of existential variables in Q exist
3     $D_{copy} := D^{\mathcal{Q}}$
4     eliminate the variables $x_{(i+1,1)}, ..., x_{(i+1,k(i))}, .., x_{(t,1)}, ..., x_{(t,k(t))}$ of the variable
      blocks $i+1, ..., t$ in $D_{copy}$, with the help of Fourier-Motzkin elimination (FME),
      and store the new inequalities in $D_i^{\mathcal{Q}}$
5     choose the $i^{th}$ move $m_i$ of the universal player, i.e., delete those half of
      inequalities from $D^{\mathcal{Q}}$, which do not belong to $m_i$
6 solve $D_1^{\mathcal{Q}} \cup ... \cup D_t^{\mathcal{Q}}$
```

The elimination in line 4 does not effect the vertex-complexity of the given system in the remaining dimensions. Therefore, the row-complexity of D_i is $\phi_{D_i} \leq 3\nu_D n^2 \leq 3 \cdot 4\phi_D n^2 \cdot n^2$. The deletions in line 5 do not increase the row-complexity of $D^{\mathcal{Q}}$. Therefore, the row-complexity of $\phi_{D_1 \cup ... \cup D_t} \leq 12\phi_D n^4$. A last application of Lemma 5 leads us to $\nu_{QLP'} = O(\nu_{D_1 \cup ... \cup D_t}) = O(\phi_D n^6) = O(\phi_{QLP'} n^6)$.

Lemma 7. *The vertex-complexity of the original QLP-polytope $P([Q : Ax \leq b])$ is of order $O(\phi_{QLP} n^6)$.*

Proof. Let two QLP-instances $\mathcal{Q}, \mathcal{Q}'$ be given, and let without loss of generality the quantifier string of $\mathcal{Q}$ end with the existential quantifiers $(q_{j+1}, ..., q_n)$, q_j being the universal quantifier with largest index. Let $\mathcal{Q}$ and $\mathcal{Q}'$ be equal except that in $\mathcal{Q}'$ the universal quantifier q_j is shifted to the end of the quantifier string, i.e., $Q' = (q_1, ..., q_{j-1}, q_{j+1}, q_j)$. Now, let us inspect the given algorithm, acting on $\mathcal{Q}'$ and $\mathcal{Q}$. Assume that the sets $D_1^{\mathcal{Q}'} \cup ... \cup D_t^{\mathcal{Q}'}$ are already known. Moving the quantifier q_j from the last position to the position j in $\mathcal{Q}'$ has the following effect. In $\mathcal{Q}'$ the existential player must fix the variables $x_{j+1}, ..., x_n$ before the universal player decides x_j. In $\mathcal{Q}$, this is not the case, and in order to express that the existential player has a choice after x_j is fixed, the variables $x_{j+1}, ..., x_n$ must be duplicated in the representation of the corresponding DEP, let us say to ${x'}_{j+1}, ..., {x'}_n$ (cf. Fig. 4). This increases the dimension of the deterministic equivalent and, at the same time, partitions the inequalities belonging to different scenarios, one induced by $x_j = 0$ and the other induced by $x_j = 1$, whereas x_j is a universal variable. Thus, due to the nature of the Fourier-Motzkin elimination, every new inequality in $D_i^{\mathcal{Q}}$, processed by line 4 of the above algorithm, is also contained in $D_i^{\mathcal{Q}'}$ as well. Inductively, it follows that $D_1^{\mathcal{Q}} \cup ... \cup D_t^{\mathcal{Q}}$ is a subset of $D_1^{\mathcal{Q}'} \cup ... \cup D_t^{\mathcal{Q}'}$, and therefore with $\mathcal{Q}' := QLP'$ of Lemma 6: $\phi_{D_1^{\mathcal{Q}} \cup ... \cup D_t^{\mathcal{Q}}} \leq \phi_{D_1^{\mathcal{Q}'} \cup ... \cup D_t^{\mathcal{Q}'}}$ and thus $\nu_{QLP} = O(\phi_{D_1^{\mathcal{Q}} \cup ... \cup D_t^{\mathcal{Q}}} n^6) = O(\phi_{D_1^{\mathcal{Q}'} \cup ... \cup D_t^{\mathcal{Q}'}} n^6) = O(\phi_{QLP} n^6)$. Thus, Theorem 2 is proven as well.

4 Algorithms Using Polynomial Space

The task of the following algorithm is to find suitable first-stage variables or the information that no solution exists. Because single paths of extremal strategies can be expressed with $O(\phi n^6)$ bits, it is possible to find a winning strategy, if it exists, with the help of a depth first search (DFS). On each level of the DFS, one of the variables is fixed, and the choices on each level are all numbers of $\mathbb{Q}$ that can be encoded with $O(\phi n^6)$ many bits. This technique can be applied to the integer version of the QLP problem and to the mixed integer version, as well.

Another, more practical way, to organize the solution process in polynomial space is based upon the nested decomposition algorithm (NDA). Nested decomposition is based on the observation that the corresponding implicit scenario tree can be transformed to a DEP that can be solved by a recursive application of Benders decomposition To solve a QLP with this algorithm, we construct a DEP as well. However, instead of encoding the scenario tree, we encode the decision tree of the universal player into the deterministic equivalent by replicating the inequalities of the QLP-instance for each possible decision combination of the

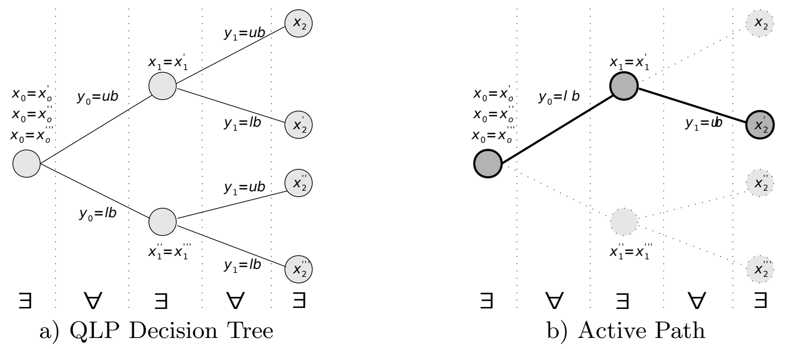


Fig. 5. QLP Decision Tree and Active Path

universal player. A mapping from a QLP with the quantifier sequence ∃∀∃∀∃ to a decision tree is shown in Figure 5a). Due to the nature of QLPs, we can use the following simplifying assumptions for our variant of nested benders. First of all, since QLPs do not have an objective function, we only need to pass back Bender's feasibility cuts to parent nodes. Furthermore, because we only need to consider the upper and lower bound of each universally quantified variable, each node only has two successors. Since we only need a variable allocation for the first-stage variables, the only type of information that has to be stored in the tree, are feasibility cuts and the proposal for the current x-vector allocation, which then belong to a specific path as depicted in Figure 5b). As described in [1] we traverse the tree *depth-first* instead *breath-first*. Thus, only nodes that are contained in the current path are *active nodes* and must therefore store cuts from their successors. When a node becomes a *passive node*, all cuts that are stored at the node can be removed, and recomputed later if needed.

To solve QLPs with polynomial space, we propose the following modification of the (implicit) decision tree, whose stages until now were determined by the quantifier changes, respectively the different quantifier blocks of existentially quantified variables. We change to a formulation where each stage consists of nodes where each master problem only corresponds to a single existential variable. The resulting decision tree now consists of nodes that either have two outgoing arcs in the case when the existentially quantified variable is the last one in the respective quantifier block, or one outgoing arc for inner variables of a block, since the successor node is not affected by an universal variable. The feasibility cuts that are passed back to the nodes at higher stages can be viewed as variable bounds. This implies that using simple preprocessing techniques each node of the current path will consist of at most two constraints, an upper and a lower bound. The polynomial space complexity directly follows.

References

1. Altenstedt, F.: Memory consumption versus computational time in nested benders decomposition for stochastic linear programming. Tech. Rep., Chalmers University, Goteborg (2003)

2. Bellmann, R.: Dynamic programming. Princeton University Press, Princeton (1957)
3. Birge, J.R., Louveaux, F.: Intro. to Stochastic Programming. Springer, Heidelberg (1997)
4. Donninger, C., Kure, A., Lorenz, U.: Parallel brutus: The first distributed, fpga accelarated chess program. In: Proc. of 18th International Parallel & Distributed Processing Symposium (IPDPS), Santa Fe. IEEE Computer Society, Los Alamitos (2004)
5. Dyer, M.E., Stougie, L.: Computational complexity of stochastic programming problems. Math. Program. 106(3), 423–432 (2006)
6. Eisenbrand, F., Rothvoß, T.: A ptas for static priority real-time scheduling with resource augmentation. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 246–257. Springer, Heidelberg (2008)
7. Engell, S., Märkert, A.M., Sand, G., Schultz, R.: Aggregated scheduling of a multiproduct batch plant by two-stage stochastic integer programming. Optimiz. and Engineering 5 (2004)
8. Grothklags, S., Lorenz, U., Monien, B.: From state-of-the-art static fleet assignment to flexible stochastic planning of the future. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 140–165. Springer, Heidelberg (2009)
9. Grötschel, M., Lovasz, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization, 2nd edn. Springer, Heidelberg (1993)
10. Hochbaum, D.S.: Approximation Algorithms for NP-hard Problems. PWS (1997)
11. Kleywegt, A.J., Shapiro, A., Homem-De-Mello, T.: The sample average approximation method for stochastic discrete optimization. SIAM Jour. of Opt., 479–502 (2001)
12. König, F.G., Lübbecke, M.E., Möhring, R.H., Schäfer, G., Spenke, I.: Solutions to real-world instances of pspace-complete stacking. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 729–740. Springer, Heidelberg (2007)
13. Liebchen, C., Lübbecke, M.E., Möhring, R.H., Stiller, S.: The concept of recoverable robustness, linear programming recovery, and railway applications. In: Robust and online large-scale optimization, pp. 1–27 (2009)
14. Megow, N., Vredeveld, T.: Approximation results for preemtive stochastic online scheduling. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 516–527. Springer, Heidelberg (2006)
15. Möhring, R.H., Schulz, A.S., Uetz, M.: Approximation in stochastic scheduling: The power of lp-based priority schedules. Journal of ACM 46(6), 924–942 (1999)
16. Papadimitriou, C.H.: Games against nature. J. of Comp. and Sys. Sc, 288–301 (1985)
17. Pochet, Y., Wolsey, L.A.: Production planning by mixed integer programming. Springer Series in Operations Research and Financial Engineering. Springer, New York (2006)
18. Schultz, R.: Stochastic programming with integer variables. Math. Progr. 97, 285–309 (2003)
19. Shmoys, D.B., Swamy, C.: Stochastic optimization is (almost) as easy as deterministic optimization. In: Proc. FOCS 2004, pp. 228–237 (2004)
20. Subramani, K.: Analyzing selected quantified integer programs. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS (LNAI), vol. 3097, pp. 342–356. Springer, Heidelberg (2004)
21. Subramani, K.: On a decision procedure for quantified linear programs. Annals of Mathematics and Artificial Intelligence 51(1), 55–77 (2007)
22. Zhang, L., Hermanns, H., Eisenbrand, F., Jansen, D.N.: Flow faster: Efficient decision algorithms for probabilistic simulations. Logical Methods in Computer Science 4(4) (2008)

Approximating Parameterized Convex Optimization Problems[⋆]

Joachim Giesen[1], Martin Jaggi[2], and Sören Laue[1]

[1] Friedrich-Schiller-Universität Jena, Germany
[2] ETH Zürich, Switzerland

Abstract. We extend Clarkson's framework by considering parameterized convex optimization problems over the unit simplex, that depend on one parameter. We provide a simple and efficient scheme for maintaining an ε-approximate solution (and a corresponding ε-coreset) along the entire parameter path. We prove correctness and optimality of the method. Practically relevant instances of the abstract parameterized optimization problem are for example regularization paths of support vector machines, multiple kernel learning, and minimum enclosing balls of moving points.

1 Introduction

We study convex optimization problems over the unit simplex that are parameterized by a single parameter. We are interested in optimal solutions of the optimization problem for all parameter values, i.e., the whole solution path in the parameter. Since the complexity of the exact solution path might be exponential in the size of the input [7], we consider approximate solutions with an approximation guarantee for all parameter values, i.e., approximate solutions along the whole path. We provide a general framework for computing approximate solution paths that has the following properties:

(1) *Generality.* Apart from being specified over the unit simplex, we hardly make any assumptions on the optimization problem under consideration. Hence, the framework can be applied in many different situations.
(2) *Simplicity.* The basic idea behind the framework is a very simple continuity argument.
(3) *Practicality.* It has been shown that our framework works well for real world problems.
(4) *Efficiency.* Although the framework is very simple it still gives improved theoretical bounds for known problems.
(5) *Optimality.* We show that it is the best possible one can do up to a constant factor.

[⋆] This work has been supported by the DFG under grant GI-711/3-1, by a Google Research Award, and by the Swiss National Science Foundation (SNF Project 20PA21-121957).

Let us explain the different aspects in more detail.

Generality: We build on the general primal-dual approximation criterion that has been introduced by Clarkson in his coreset framework [5] for convex optimization problems over the unit simplex. Among the many problems that fit into Clarkson's framework are for example the smallest enclosing ball problem, polytope distance problems, and binary classification support vector machines. For many of these problems parameterized versions are known, e.g., the smallest enclosing ball problem for points that move with time, or soft margin support vector machines that trade-off a regularization term and a loss term in the objective function of the corresponding optimization problem.

Simplicity: The basic algorithmic idea behind our framework is computing at some parameter value an approximate solution whose approximation guarantee holds for some sub-interval of the problem path. This solution is then updated at the boundary of the sub-interval to a better approximation that remains a good approximation for a consecutive sub-interval. For computing the initial approximation and the updates from previous approximations, any arbitrary (possibly problem specific) algorithm can be used, that ideally can be started from the previous solution (warm start). We provide a simple lemma that allows to bound the number of necessary parameter sub-intervals for a prescribed approximation quality. For interesting problems, the lemma also implies the existence of small coresets that are valid for the entire parameter path.

Practicality: Our work is motivated by several problems from machine learning and computational geometry that fit into the described framework, in particular, support vector machines and related classification methods, multiple kernel learning [3], and the smallest enclosing ball problem [4]. We have implemented the proposed algorithms and applied them to choose the optimal regularization parameter for a support vector machine, and to find the best combination of two kernels which is a special case of the multiple kernel learning problem.

Efficiency: Our framework gives a path complexity of $O\left(\frac{1}{\varepsilon}\right)$, meaning that an ε-approximate solution needs to be updated only $O\left(\frac{1}{\varepsilon}\right)$ times along the whole path. This positively contrasts the complexity of exact solution paths.

Optimality: We provide lower bounds that show that one cannot do better, i.e., there exist examples where one needs essentially at least one fourth as many sub-intervals as predicted by our method.

Related Work. Many of the aforementioned problems have been recently studied intensively, especially machine learning methods such as computing exact solution paths in the context of support vector machines and related problems [10,12,13,8]. But exact algorithms can be fairly slow compared to approximate methods. To make things even worse, the complexity of exact solution paths can be very large, e.g., it can grow exponentially in the input size as it has been shown for support vector machines with ℓ_1-loss term [7]. Hence, approximation algorithms have become popular also for the case of solution paths lately, see e.g. [6]. However, to the best of our knowledge, so far no approximation quality guarantees along the path could be given for any of these existing algorithms.

2 Clarkson's Framework

In [5] Clarkson considers convex optimization problems of the form

$$\begin{array}{ll} \min_x f(x) \\ \text{s.t.} \quad x \in S_n \end{array} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is convex and continuously differentiable, and S_n is the unit simplex, i.e., S_n is the convex hull of the standard basis vectors of $\mathbb{R}^n$. We additionally assume that the function f is non-negative on S_n. A point $x \in \mathbb{R}^n$ is called a feasible solution, if $x \in S_n$.

The Lagrangian dual of Problem 1 (sometimes also called Wolfe dual) is given by the unconstrained problem

$$\max_x \omega(x), \text{ where } \omega(x) := f(x) + \min_i (\nabla f(x))_i - x^T \nabla f(x).$$

In this framework Clarkson studies approximating the optimal solution. His measure of approximation quality is (up to a multiplicative positive constant) the primal-dual gap

$$g(x) := f(x) - \omega(x) = x^T \nabla f(x) - \min_i (\nabla f(x))_i.$$

Note that convexity of f implies the weak duality condition $f(\hat{x}) \geq \omega(x)$, for the optimal solution $\hat{x} \in S_n$ of the primal problem and any feasible solution x, which in turn implies non-negativity of the primal-dual gap, i.e., $g(x) \geq 0$ for all feasible x, see [5]. A feasible solution x is an *ε-approximation* to Problem 1 if

$$g(x) \leq \varepsilon f(x).$$

Sometimes an ε-approximation is defined more restrictively as $g(x) \leq \varepsilon f(\hat{x})$, relative to the optimal value $f(\hat{x})$ of the primal optimization problem. Note that this can directly be obtained from our slightly weaker definition by setting ε in the definition of an ε-approximation to $\varepsilon' := \frac{\varepsilon}{1+\varepsilon}$, because $g(x) \leq \frac{\varepsilon}{1+\varepsilon} f(x)$ $\Leftrightarrow (1+\varepsilon)(f(x) - \omega(x)) \leq \varepsilon f(x) \Leftrightarrow g(x) \leq \varepsilon \omega(x) \leq \varepsilon f(\hat{x})$. A subset $C \subseteq [n]$ is called an *ε-coreset*, if there exists an ε-approximation x to Problem 1 with $x_i = 0, \forall i \in [n] \setminus C$.

The case of *maximizing* a concave, continuously differentiable, non-negative function f over the unit simplex S_n can be treated analogously. The Lagrangian dual problem is given as

$$\min_x \omega(x), \text{ where } \omega(x) := f(x) + \max_i (\nabla f(x))_i - x^T \nabla f(x),$$

and the duality gap is $g(x) := \omega(x) - f(x) = \max_i (\nabla f(x))_i - x^T \nabla f(x)$. Again, $x \in S_n$ is an ε-approximation if $g(x) \leq \varepsilon f(x)$ (which immediately implies $g(x) \leq \varepsilon f(\hat{x})$ for the optimal solution $\hat{x}$ of the primal maximization problem).

Clarkson [5] showed that ε-coresets of size $\left\lceil \frac{2C_f}{\varepsilon} \right\rceil$ do always exist, and that the sparse greedy algorithm [5, Algorithm 1.1] obtains an ε-approximation after at most $2 \left\lceil \frac{4C_f}{\varepsilon} \right\rceil$ many steps. Here C_f is an absolute constant describing the "non-linearity" or "curvature" of the function f.

3 Optimizing Parameterized Functions

We extend Clarkson's framework and consider parameterized families of functions $f_t(x) = f(x;t) : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ that are convex and continuously differentiable in x and parameterized by $t \in \mathbb{R}$, i.e., we consider the following families of minimization problems

$$\begin{array}{ll} \min_x & f_t(x) \\ \text{s.t.} & x \in S_n \end{array} \tag{2}$$

Again, we assume $f_t(x) \geq 0$ for all $x \in S_n$ and $t \in \mathbb{R}$.

The following simple lemma is at the core of our discussion and characterizes how we can change the parameter t such that a given $\frac{\varepsilon}{\gamma}$-approximate solution x (for $\gamma > 1$) at t stays an ε-approximate solution.

Lemma 1. *Let $x \in S_n$ be an $\frac{\varepsilon}{\gamma}$-approximation to Problem 2 for some fixed parameter value t, and for some $\gamma > 1$. Then for all $t' \in \mathbb{R}$ that satisfy*

$$\begin{aligned} &x^T \nabla(f_{t'}(x) - f_t(x)) - (\nabla(f_{t'}(x) - f_t(x)))_i - \varepsilon(f_{t'}(x) - f_t(x)) \\ &\quad \leq \varepsilon\left(1 - \tfrac{1}{\gamma}\right) f_t(x), \qquad \forall i \in [n], \end{aligned} \tag{3}$$

the solution x is still an ε-approximation to Problem 2 at the changed parameter value t'.

Proof. We have to show that $g(x;t') \leq \varepsilon f_{t'}(x)$, or in other words that

$$x^T \nabla f_{t'}(x) - (\nabla f_{t'}(x))_i \leq \varepsilon f_{t'}(x)$$

holds for all components i. We add to the Inequalities 3 for all components i the inequalities stating that x is an $\frac{\varepsilon}{\gamma}$-approximate solution at value t, i.e.

$$x^T \nabla f_t(x) - (\nabla f_t(x))_i \leq \frac{\varepsilon}{\gamma} f_t(x).$$

This gives for all $i \in [n]$

$$x^T \nabla f_{t'}(x) - (\nabla f_{t'}(x))_i - \varepsilon(f_{t'}(x) - f_t(x)) \leq \varepsilon f_t(x),$$

which simplifies to the claimed bound $x^T \nabla f_{t'}(x) - (\nabla f_{t'}(x))_i \leq \varepsilon f_{t'}(x)$. □

The analogue of Lemma 1 for *maximizing* a concave function over the unit simplex is the following lemma whose proof follows along the same lines:

Lemma 2. *Let $x \in S_n$ be an $\frac{\varepsilon}{\gamma}$-approximation to the maximization problem $\max_{x \in S_n} f_t(x)$ at parameter value t, for some $\gamma > 1$. Here $f_t(x)$ is a parameterized family of concave, continuously differentiable functions in x that are non-negative on S_n. Then for all $t' \in \mathbb{R}$ that satisfy*

$$\begin{aligned} &(\nabla(f_{t'}(x) - f_t(x)))_i - x^T \nabla(f_{t'}(x) - f_t(x)) - \varepsilon(f_{t'}(x) - f_t(x)) \\ &\quad \leq \varepsilon\left(1 - \tfrac{1}{\gamma}\right) f_t(x), \qquad \forall i \in [n], \end{aligned} \tag{4}$$

the solution x is still an ε-approximation at the changed parameter value t'.

We define the *ε-approximation path complexity* for Problem 2 as the minimum number of sub-intervals of the parameter interval $\mathbb{R}$ such that for each sub-interval there is a single solution of Problem 2 which is an ε-approximation for that entire sub-interval.

Lemma 1 and 2 imply upper bounds on the path complexity. Next, we will show that these upper bounds are tight up to a multiplicative factor of $4 + 2\varepsilon$.

3.1 Lower Bound

To see that the approximate path complexity bounds we get from Lemma 2 are optimal consider the following parameterized optimization problem:

$$\begin{array}{ll} \max_x & f_t(x) := x^T f(t) \\ \text{s.t.} & x \in S_n \end{array} \tag{5}$$

where $f(t) = (f_0(t), \ldots, f_k(t))$ is a vector of functions and $f_i(t)$ is defined as follows

$$f_i(t) = \begin{cases} 0, & \text{for } t < i\varepsilon' \\ t - i\varepsilon', & \text{for } i\varepsilon' \le t < 1 + i\varepsilon' \\ -t + 2 + i\varepsilon', & \text{for } 1 + i\varepsilon' \le t \le 2 + i\varepsilon' \\ 0, & \text{for } 2 + i\varepsilon' < t \end{cases}$$

for some arbitrary fixed $\varepsilon' > 0$ and $n \ge 1/\varepsilon'$. See Figure 1 for an illustration of the function $f_i(t)$.

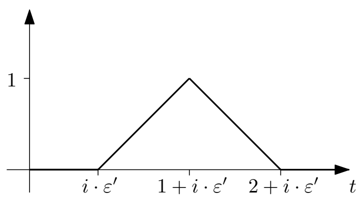


Fig. 1. Function $f_i(t)$

Each of the $f_i(t)$ attains its maximum 1 at $t = 1 + i\varepsilon'$. Since $f_t(x)$ is linear in x it is hence concave in x for every fixed t. Hence, it is an instance of Problem 2. Let us now consider the interval $t \in [1, 2]$. In this interval consider the points $t_i := 1 + i\varepsilon'$, for $i = 0, \ldots, \lfloor 1/\varepsilon' \rfloor$. At each of these points it holds that $f_i(t_i) = 1$ and all the other $f_j(t_i) \le 1 - \varepsilon'$ when $j \ne i$. Hence, the value of the optimal solution to Problem 5 at parameter value t_i is 1, and it is attained at $x = e_i$, where e_i is the i-th standard basis vector. Furthermore, for all other $x \in S_n$ that have an entry at the coordinate position i that is at most 1/2 it holds that $f_{t_i}(x) \le 1 - \varepsilon'/2$.

Hence, in order to have an ε-approximation for $\varepsilon < \varepsilon'/2$ the approximate solution x needs to have an entry of more than 1/2 at the i-th coordinate position. Since all entries of x sum up to 1, all the other entries are strictly less than 1/2 and hence this solution cannot be an ε-approximation for any other parameter value $t = t_j$ with $j \ne i$. Thus, for all values of $t \in [1, 2]$ one needs at least $1/\varepsilon'$ different solutions for any $\varepsilon < \varepsilon'/2$.

Choosing ε' arbitrarily close to 2ε this implies that one needs at least $\frac{1}{2\varepsilon} - 1$ different solutions to cover the whole path for $t \in [1, 2]$.

Lemma 2 gives an upper bound of $\frac{2+\varepsilon}{\varepsilon}\frac{\gamma}{\gamma-1} = \left(\frac{2}{\varepsilon} + 1\right)\frac{\gamma}{\gamma-1}$ different solutions, since $\nabla f_t(x) = f(t) = (f_i(t))_{i \in [n]}$ and $\left|\frac{\partial f_i}{\partial t}\right| \le 1$, $\left(\nabla(f_{t'}(x) - f_t(x))\right)_i \le |t' - t|$.

Hence, this is optimal up to a factor of $4+2\varepsilon$. Indeed, also the dependence on the problem specific constants in Lemma 2 is tight: 'contracting' the functions $f_i(t)$ along the t-direction increases the Lipschitz constant of $(\nabla f_t(x))_i$, which can be shown to be an upper bound on the problem specific constants in Lemma 2.

3.2 The Weighted Sum of Two Convex Functions

We are particularly interested in a special case of Problem 2. For any two convex, continuously differentiable functions $f^{(1)}, f^{(2)} : \mathbb{R}^n \to \mathbb{R}$ that are non-negative on S_n, we consider the weighted sum $f_t(x) := f^{(1)}(x)+tf^{(2)}(x)$ for a real parameter $t \geq 0$. The parameterized optimization Problem 2 in this case becomes:

$$\begin{array}{l} \min_x \; f^{(1)}(x) + tf^{(2)}(x) \\ \text{s.t.} \quad x \in S_n \end{array} \tag{6}$$

For this optimization problem we have the following corollary of Lemma 1:

Corollary 1. *Let $x \in S_n$ be an $\frac{\varepsilon}{\gamma}$-approximate solution to Problem 6 for some fixed parameter value $t \geq 0$, and for some $\gamma > 1$. Then for all $t' \geq 0$ that satisfy*

$$(t'-t)\left(x^T\nabla f^{(2)}(x) - (\nabla f^{(2)}(x))_i - \varepsilon f^{(2)}(x)\right) \leq \varepsilon\left(1-\frac{1}{\gamma}\right) f_t(x), \quad \forall i \in [n] \tag{7}$$

solution x is an ε-approximate solution to Problem 6 at the parameter value t'.

Proof. Follows directly from Lemma 1, and $f_{t'}(x) - f_t(x) = (t'-t)f^{(2)}(x)$. □

This allows us to determine the entire interval of admissible parameter values t' such that an $\frac{\varepsilon}{\gamma}$-approximate solution at t is still an ε-approximate solution at t'.

Corollary 2. *Let x be an $\frac{\varepsilon}{\gamma}$-approximate solution to the Problem 6 for some fixed parameter value $t \geq 0$, for some $\gamma > 1$, and let*

$$u := x^T\nabla f^{(2)}(x) - \min_i \left(\nabla f^{(2)}(x)\right)_i - \varepsilon f^{(2)}(x)$$
$$l := x^T\nabla f^{(2)}(x) - \max_i \left(\nabla f^{(2)}(x)\right)_i - \varepsilon f^{(2)}(x),$$

then x remains an ε-approximate solution for all $0 \leq t' = t + \delta$ for the following values of δ:

(i) If $l < 0$ or $0 < u$, then the respective admissible values for δ are

$$\varepsilon\left(1-\frac{1}{\gamma}\right)\frac{f_t(x)}{l} \leq \delta \leq \varepsilon\left(1-\frac{1}{\gamma}\right)\frac{f_t(x)}{u}$$

(ii) If $u \leq 0$, then δ (and thus t') can become arbitrarily large.
(iii) If $l \geq 0$, then δ can become as small as $-t$, and thus t' can become 0.

Note that the ε-approximation path complexity for Problem 6 for a given value of $\gamma > 1$ can be upper bounded by the minimum number of points $t_j \geq 0$ such that the admissible intervals of $\frac{\varepsilon}{\gamma}$-approximate solutions x_j at t_j cover the whole parameter interval $[0, \infty)$.

Corollary 2 immediately suggests two variants of an algorithmic framework (forward- and backwards version) maintaining ε-approximate solutions over the entire parameter interval or in other words, tracking a guaranteed ε-approximate solution path. Note that as the internal optimizer, any arbitrary approximation algorithm can be used here, as long as it provides an approximation guarantee on the relative primal-dual gap. For example the standard Frank-Wolfe algorithm [5, Algorithm 1.1] is particularly suitable as its resulting coreset solutions are also sparse. The forward version is depicted in Algorithm 1.

Algorithm 1. ApproximationPath—ForwardVersion $(\varepsilon, \gamma, t_{\min}, t_{\max})$

```
1  compute an $\frac{\varepsilon}{\gamma}$-approximation $x$ for $f_t(x)$ at $t := t_{\min}$ using a standard optimizer.
2  do
3      $u := x^T \nabla f^{(2)}(x) - \min_i \left(\nabla f^{(2)}(x)\right)_i - \varepsilon f^{(2)}(x)$
4      if $u > 0$ then
5          $\delta := \varepsilon \left(1 - \frac{1}{\gamma}\right) \frac{f_t(x)}{u} > 0$
6          $t := t + \delta$
7          improve the (now still $\varepsilon$-approximate) solution $x$ for $f_t(x)$ to an at least
           $\frac{\varepsilon}{\gamma}$-approximate solution by applying steps of any standard optimizer.
8      else
9          $t := t_{\max}$
10 while $t < t_{\max}$
```

4 Applications

Special cases of Problem 6 or the more general Problem 2 have applications in computational geometry and machine learning. In the following we discuss two applications in more detail, namely, a parameterized polytope distance problem as an application of the special case of Problem 6, and smallest enclosing balls of linearly moving points as an application of the general Problem 2. The polytope distance problem itself can be specialized for computing regularization paths of support vector machines (SVMs), and for kernel learning in the context of SVMs.

4.1 A Parameterized Polytope Distance Problem

In the setting of Section 3.2 we consider the case $f^{(1)}(x) := x^T K^{(1)} x$ and $f^{(2)}(x) := x^T K^{(2)} x$, for two positive semi-definite matrices $K^{(1)}, K^{(2)} \in \mathbb{R}^{n \times n}$, i.e.,

$$\begin{array}{ll} \min_x & f^{(1)}(x) + t f^{(2)}(x) = x^T \left(K^{(1)} + t K^{(2)}\right) x \\ \text{s.t.} & x \in S_n \ . \end{array} \tag{8}$$

The geometric interpretation of this problem is as follows: let $A(t) \in \mathbb{R}^{n \times r}$, $r \leq n$, be the unique matrix such $A(t)^T A(t) = K^{(1)} + tK^{(2)}$ (Cholesky decomposition). The solution $\hat{x}$ to Problem 8 is the point in the convex hull of the column vectors of the matrix $A(t)$ that is closest to the origin. Hence, Problem 8 is a parameterized polytope distance problem. For the geometric interpretation of an ε-approximation in this context we refer to [9]. In the following we will always assume that the two original polytope distance problems are separable, i.e. $\min_{x \in S_n} x^T K^{(1)} x > 0$ and $\min_{x \in S_n} x^T K^{(2)} x > 0$.

For this parameterized problem, the two quantities u and l that determine the admissible parameter intervals in Corollary 2 and the step size in both approximate path algorithms take the simpler form

$$u = (2-\varepsilon) x^T K^{(2)} x - 2 \min_i (K^{(2)} x)_i \text{ and } l = (2-\varepsilon) x^T K^{(2)} x - 2 \max_i (K^{(2)} x)_i,$$

since $\nabla f^{(2)}(x) = 2K^{(2)} x$. We now use the following lemma to bound the path complexity for instances of Problem 8.

Lemma 3. *Let $0 < \varepsilon \leq 1$ and $\gamma > 1$. Then for any parameter $t \geq 0$, the length of the interval $[t-\delta, t]$ with $\delta > 0$, on which an $\frac{\varepsilon}{\gamma}$-approximate solution x to Problem 8 at parameter value t remains an ε-approximation, is at least*

$$l_f(\varepsilon, \gamma) := \frac{\varepsilon}{2} \left(1 - \frac{1}{\gamma}\right) \frac{\min\limits_{x \in S_n} x^T K^{(1)} x}{\max\limits_{x \in S_n} x^T K^{(2)} x} = \Omega(\varepsilon) \ . \tag{9}$$

Proof. For $l = (2-\varepsilon) x^T K^{(2)} x - 2 \max_i (K^{(2)} x)_i < 0$, we get from Corollary 2 that the length of the left interval at x is of length at least

$$\varepsilon \left(1 - \frac{1}{\gamma}\right) \frac{f_t(x)}{-l}.$$

For any $t \geq 0$, we can lower bound

$$f_t(x) \geq f^{(1)}(x) = x^T K^{(1)} x \geq \min_{x \in S_n} x^T K^{(1)} x,$$

and for $\varepsilon \leq 1$ we can upper bound

$$-l = 2 \max_i (K^{(2)} x)_i - (2-\varepsilon) x^T K^{(2)} x \leq 2 \max_i (K^{(2)} x)_i,$$

because $f^{(2)}(x) \geq 0$. The value $\max_i (K^{(2)} x)_i = \max_i e_i^T K^{(2)} x$ is the inner product between two points in the convex hull of the columns of the square root of the positive semi-definite matrix $K^{(2)}$ (see the discussion at the beginning of this section). Let these two points be $u, v \in \mathbb{R}^n$. Using the Cauchy-Schwarz inequality we get

$$\begin{aligned} \max_i (K^{(2)} x)_i = u^T v &\leq \sqrt{\|u\|^2 \|v\|^2} \leq \frac{1}{2} (\|u\|^2 + \|v\|^2) \\ &\leq \max\{\|u\|^2, \|v\|^2\} \leq \max_{x \in S_n} x^T K^{(2)} x, \end{aligned}$$

where the last expression gives the norm of the largest vector with endpoint in the convex hull of the columns of the square root of $K^{(2)}$. Hence, $-l \leq 2\max_{x\in S_n} x^T K^{(2)} x$. Combining the lower bound for $f_t(x)$ and the upper bound for $-l$ gives the stated bound on the interval length. $\square$

Now, to upper bound the approximation path complexity we split the interval $[0,\infty]$ into two parts: the sub-interval $[0,1]$ can be covered by at most $1/l_f(\varepsilon,\gamma)$ admissible left intervals, i.e., by at most $1/l_f(\varepsilon,\gamma)$ many admissible intervals. We reduce the analysis for the sub-interval $t \in [1,\infty]$ to the analysis for $[0,1]$ by interchanging the roles of $f^{(1)}$ and $f^{(2)}$. For any $t \geq 1$, x is an ε-approximate solution to $\min_{x\in S_n} f_t(x) := f^{(1)}(x) + tf^{(2)}(x)$ if and only if x is an ε-approximate solution to $\min_{x\in S_n} f'_{t'}(x) := t'f^{(1)}(x) + f^{(2)}(x)$ for $t' = \frac{1}{t} \leq 1$, because the definition of an ε-approximation is invariant under scaling the objective function. Note that by allowing $t = \infty$ we just refer to the case $t' = 0$ in the equivalent problem for $f'_{t'}(x)$ with $t' = \frac{1}{t} \in [0,1]$. Using the lower bounds on the interval lengths $l_f(\varepsilon,\gamma)$ and $l_{f'}(\varepsilon,\gamma)$ (for the problem for $f'_{t'}(x)$ with $t' \in [0,1]$) on both sub-intervals we get an upper bound of $\left\lceil \frac{1}{l_f(\varepsilon,\gamma)} \right\rceil + \left\lceil \frac{1}{l_{f'}(\varepsilon,\gamma)} \right\rceil$ on the path complexity as is detailed in the following theorem:

Theorem 1. *Given any $0 < \varepsilon \leq 1$ and $\gamma > 1$, the ε-approximation path complexity of Problem 8 is at most*

$$\frac{\gamma}{\gamma-1}\left(\frac{\max\limits_{x\in S_n} x^T K^{(2)} x}{\min\limits_{x\in S_n} x^T K^{(1)} x} + \frac{\max\limits_{x\in S_n} x^T K^{(1)} x}{\min\limits_{x\in S_n} x^T K^{(2)} x}\right)\frac{2}{\varepsilon} + 2 = O\left(\frac{1}{\varepsilon}\right) .$$

This proof of the path complexity immediately implies a bound on the time complexity of our approximation path Algorithm 1. In particular we obtain a linear running time of $O\left(\frac{n}{\varepsilon^2}\right)$ for computing the global solution path when using [5, Algorithm 1.1] as the internal optimizer.

There are interesting applications of this result, because it is known that instances of Problem 8 include for example computing the solution path of a support vector machine – as the regularization parameter changes – and also finding the optimal combination of two kernel matrices in the setting of kernel learning. For more details we refer the reader to the full version of this paper.

4.2 Minimum Enclosing Ball of Points under Linear Motion

Let $P = \{p_1, \dots, p_n\}$ be a set of n points in $\mathbb{R}^d$. The minimum enclosing ball (MEB) problem asks to find the smallest ball containing all points of P. The dual of the problem can be written [11] as:

$$\begin{array}{ll} \max_x & x^T b - x^T A^T A x \\ \text{s.t.} & x \in S_n \end{array} \tag{10}$$

where $b = (b_i) = (p_i^T p_i)_{i\in[n]}$ and A is the matrix whose columns are the p_i.

Now we assume that the points move with constant speed in a fixed direction, i.e., they move linearly as follows

$$p_i(t) = p_i + tv_i, \quad t \in [0, \infty)$$

where t can be referred to as time parameter. The MEB problem for moving points reads as:

$$\begin{array}{ll} \max_x \; x^T b(t) - x^T (P + tV)^T (P + tV)x \\ \text{s.t.} \quad x \in S_n \end{array} \tag{11}$$

where $b(t) = (b_i(t)) = ((p_i + tv_i)^T (p_i + tv_i))_{i \in [n]}$ and P is the matrix whose columns are the points p_i and V is the matrix whose columns are the vector v_i. Problem 11 is a special case of the maximization version of Problem 6. Again, we are interested in the whole solution path, i.e. we want to track the center and the radius

$$r(t) = \sqrt{\hat{x}^T b(t) - \hat{x}^T (P + tV)^T (P + tV)\hat{x}} \qquad \text{with } \hat{x} \in S_n \text{ optimal}$$

of the MEB of the points $p_i(t)$ for $t \in [0, \infty)$ (or approximations of it). For an analysis of an approximate solution path we make use of the following observation.

Observation 1. *The interval* $[0, \infty)$ *can be subdivided into three parts: on the first sub-interval* $r(t)$ *is decreasing, on the second sub-interval, the radius* $r(t)$ *is constant, and on the third sub-interval, the radius is increasing.*

This can be seen as follows: consider the time when the radius of the MEB reaches its global minimum, just before the ball is expanding again. This is the point between the second and the third sub-interval. The points that cause the ball to expand at this point in time will prevent the ball from shrinking again in the future since the points move linearly. Thus the radius of the MEB will increase on the third sub-interval. By reversing the direction of time the same consideration leads to the observation that the radius of the MEB is decreasing on the first sub-interval.

We will consider each of the three sub-intervals individually. The second sub-interval can be treated like the standard MEB problem of non-moving points. Hence we only have to consider the first and the third sub-interval. We will only analyze the third sub-interval since the first sub-interval can be treated analogously with the direction of time reversed, i.e., the parameter t decreasing instead of increasing.

For the third sub-interval we know that the radius is increasing with time. We can shift the time parameter t such that we start with the third sub-interval at time $t = 0$. Let $r > 0$ be the radius $r(0)$ at time zero, i.e., we assume that the radius of the MEB never becomes zero. The case where the radius reaches 0 at some point is actually equivalent to the standard MEB problem for non-moving points. Without loss of generality we can scale all the vectors v_i such that the MEB defined by the points v_i has radius r as well, because this just means scaling time. Without loss of generality we can also assume that the center of the MEB of the point sets P and V are both the origin. That is, $\|p_i\| \leq r$ and $\|v_i\| \leq r$. We have $f_t(x) := x^T b(t) - x^T (P + tV)^T (P + tV)x$. A short computation shows that

$$(\nabla f_{t+\delta}(x) - \nabla f_t(x))_i - x^T(\nabla f_{t+\delta}(x)) - \nabla f_t(x)) \leq 12r^2(1 + t + \delta)\delta.$$

and

$$|f_{t+\delta}(x) - f_t(x)| \leq 4r^2(1 + t + \delta)\delta.$$

Now we can apply Lemma 2. Inequality 4 here simplifies to

$$12r^2(1 + t + \delta)\delta + \varepsilon 4r^2(1 + t + \delta)\delta \leq \varepsilon \left(1 - \frac{1}{\gamma}\right) r^2$$

since $f_t(x) \geq r^2$. Assuming $\varepsilon \leq 1$, we can set $\delta = \frac{\varepsilon}{32}\left(1 - \frac{1}{\gamma}\right)$ for $t, t + \delta \in [0, 1]$. For the interval of $t, t + \delta = [1, \infty)$ we apply the same trick as before and reduce it to the case of $t, t + \delta \in [0, 1]$ by interchanging the roles of P and V. A short computation shows that an ε-approximation x at time $t \geq 1$ for the original optimization problem

$$\begin{array}{ll} \max_x & x^T b(t) - x^T (P + tV)^T (P + tV)x \\ \text{s.t.} & x \in S_n \end{array}$$

is an ε-approximation for the optimization problem

$$\begin{array}{ll} \max_x & x^T b'(t') - x^T (t'P + V)^T (t'P + V)x \\ \text{s.t.} & x \in S_n \end{array}$$

at time $t' = 1/t$, where $b'(t') = (b'_i(t')) = ((t'p_i + v_i)^T(t'p_i + v_i))_{i\in[n]}$, i.e., the roles of P and V have been interchanged. This is again due to the fact that the relative approximation guarantee is invariant under scaling. Hence, we conclude with the following theorem on the approximation path complexity for the minimum enclosing ball problem under linear motion:

Theorem 2. *The ε-approximation path complexity of the minimum enclosing ball Problem 11 for parameter $t \in [0, \infty)$ is at most*

$$64\frac{\gamma}{\gamma - 1}\frac{1}{\varepsilon} = O\left(\frac{1}{\varepsilon}\right).$$

Since for the static MEB Problem 10, coresets of size $O(\frac{1}{\varepsilon})$ exist, see [4], we obtain the following corollary to Theorem 2.

Corollary 3. *There exists an ε-coreset of size $O(\frac{1}{\varepsilon^2})$ for Problem 11 that is globally valid under the linear motion, i.e., valid for all $t \geq 0$.*

The only other result known in this context is the existence of coresets of size $2^{O\left(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon}\right)}$ that remain valid under polynomial motions [2], and earlier, Agarwal et al. [1] have already proven the existence of coresets of size $O(1/\varepsilon^{2d})$ for the extent problem for moving points, which includes the MEB problem as a special case.

5 Conclusion

We have presented a framework to optimize convex functions over the unit simplex that are parameterized in one parameter. The framework is very general, simple and has been proven to be practical on a number of machine learning problems. Although it is very simple it still provides improved theoretical bounds on known problems. In fact, we showed that our method is optimal up to a small constant factor.

References

1. Agarwal, P., Har-Peled, S., Varadarajan, K.: Approximating extent measures of points. Journal of the ACM 51(4), 606–635 (2004)
2. Agarwal, P., Har-Peled, S., Yu, H.: Embeddings of surfaces, curves, and moving points in euclidean space. In: SCG 2007: Proceedings of the Twenty-third Annual Symposium on Computational Geometry (2007)
3. Bach, F., Lanckriet, G., Jordan, M.: Multiple kernel learning, conic duality, and the smo algorithm. In: ICML 2004: Proceedings of the Twenty-first International Conference on Machine Learning (2004)
4. Bădoiu, M., Clarkson, K.L.: Optimal core-sets for balls. Computational Geometry: Theory and Applications 40(1), 14–22 (2007)
5. Clarkson, K.L.: Coresets, sparse greedy approximation, and the frank-wolfe algorithm. In: SODA 2008: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (2008)
6. Friedman, J., Hastie, T., Höfling, H., Tibshirani, R.: Pathwise coordinate optimization. The Annals of Applied Statistics 1(2), 302–332 (2007)
7. Gärtner, B., Giesen, J., Jaggi, M.: An exponential lower bound on the complexity of regularization paths. arXiv, cs.LG (2009)
8. Gärtner, B., Giesen, J., Jaggi, M., Welsch, T.: A combinatorial algorithm to compute regularization paths. arXiv, cs.LG (2009)
9. Gärtner, B., Jaggi, M.: Coresets for polytope distance. In: SCG 2009: Proceedings of the 25th Annual Symposium on Computational Geometry (2009)
10. Hastie, T., Rosset, S., Tibshirani, R., Zhu, J.: The entire regularization path for the support vector machine. The Journal of Machine Learning Research 5, 1391–1415 (2004)
11. Matousek, J., Gärtner, B.: Understanding and Using Linear Programming (Universitext). Springer, New York (2006)
12. Rosset, S., Zhu, J.: Piecewise linear regularized solution paths. Annals of Statistics 35(3), 1012–1030 (2007)
13. Wu, Z., Zhang, A., Li, C., Sudjianto, A.: Trace solution paths for svms via parametric quadratic programming. In: KDD 2008 DMMT Workshop (2008)

Approximation Schemes for Multi-Budgeted Independence Systems[⋆]

Fabrizio Grandoni[1] and Rico Zenklusen[2,⋆⋆]

[1] Computer Science Department, University of Rome Tor Vergata
grandoni@disp.uniroma2.it
[2] Department of Mathematics, EPFL
rico.zenklusen@epfl.ch

Abstract. A natural way to deal with multiple, partially conflicting objectives is turning all the objectives but one into budget constraints. Some classical optimization problems, such as spanning tree and forest, shortest path, (perfect) matching, independent set (basis) in a matroid or in the intersection of two matroids, become NP-hard even with one budget constraint. Still, for most of these problems efficient deterministic and randomized approximation schemes are known. For two or more budgets, typically only multi-criteria approximation schemes are available, which return slightly infeasible solutions. Not much is known however for strict budget constraints: filling this gap is the main goal of this paper.

It is not hard to see that the above-mentioned problems whose solution sets do not correspond to *independence systems* are inapproximable already for two budget constraints. For the remaining problems, we present approximation schemes for a constant number k of budget constraints using a variety of techniques: i) we present a simple and powerful mechanism to transform multi-criteria approximation schemes into *pure* approximation schemes. This leads to deterministic and randomized approximation schemes for various of the above-mentioned problems; ii) we show that points in low-dimensional faces of any matroid polytope are almost integral, an interesting result on its own. This gives a deterministic approximation scheme for k-budgeted matroid independent set; iii) we present a deterministic approximation scheme for 2-budgeted matching. The backbone of this result is a purely topological property of curves in $\mathbb{R}^2$.

1 Introduction

In many applications, one has to compromise between several, partially conflicting goals. *Multi-Objective Optimization* is a broad area of study in Operations Research, Economics and Computer Science (see [8,22] and references therein). A variety of approaches have been employed to formulate such problems. Here we

⋆ Partially developed while the first author was visiting EPFL.
⋆⋆ Partially supported by the Swiss National Science Foundation, grant number: PBEZP2-129524.

adopt the Multi-Budgeted Optimization approach [22]: we cast one of the goals as the objective function, and the others as *budget constraints*. More precisely, we are given a (finite) set $\mathcal{F}$ of solutions for the problem, where each solution is a subset S of elements from a given universe E (e.g., the edges of a graph). We are also given a weight function $\mathrm{w} : \mathcal{F} \to \mathbb{Q}_+$ and a set of $k = O(1)$[1] length functions $\ell_i : \mathcal{F} \to \mathbb{Q}_+$, $1 \leq i \leq k$, that assign a weight $\mathrm{w}(S) := \sum_{e \in S} \mathrm{w}(e)$ and an ith-length $\ell_i(S) := \sum_{e \in S} \ell_i(e)$, $1 \leq i \leq k$, to every candidate solution S. For each length function ℓ_i, there is a budget $\mathrm{L}_i \in \mathbb{Q}_+$. The *multi-budgeted optimization problem* can then be formulated as follows:

$$\text{maximize/minimize } \mathrm{w}(S) \ \text{ subject to } \ S \in \mathcal{F},\ \ell_i(S) \leq \mathrm{L}_i,\ 1 \leq i \leq k.$$

We next use OPT to denote an optimum solution.

Following the literature on the topic, we focused on the set of problems below:

- k-BUDGETED (PERFECT) MATCHING: $\mathcal{F}$ is given by the (perfect) matchings of an undirected graph $G = (V, E)$.
- k-BUDGETED SPANNING TREE (FOREST): $\mathcal{F}$ is given by the spanning trees (forests) of G.
- k-BUDGETED SHORTEST PATH: $\mathcal{F}$ is given by the paths connecting two given nodes s and t in G.
- k-BUDGETED MATROID INDEPENDENT SET (BASIS): $\mathcal{F}$ is given by the independent sets (bases) of a matroid $M = (E, \mathcal{I})$[2].
- k-BUDGETED MATROID INTERSECTION INDEPENDENT SET (BASIS): $\mathcal{F}$ is given by the independent sets (bases) in the intersection of two matroids $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$.

All the problems above are polynomial-time solvable (see, e.g., [24]) in their unbudgeted version ($k = 0$), but become NP-hard [1,3,7] even for a single budget constraint ($k = 1$). For the case of one budget ($k = 1$), polynomial-time approximation schemes (PTASs) are known for SPANNING TREE [21] (see also [11]), SHORTEST PATH [25] (see also [10,14]), and MATCHING [3]. The approach in [21] easily generalizes to the case of MATROID BASIS. A PTAS is also known for MATROID INTERSECTION INDEPENDENT SET [3]. No approximation algorithm is known for the problems above in the case $k \geq 2$ (excluding multi-criteria algorithms which provide slightly infeasible solutions): investigating the existence of such algorithms is the main goal of this paper.

1.1 Our Results

We start by observing that several of the mentioned problems are inapproximable already for two budget constraints. More precisely, the corresponding feasibility

[1] The assumption that k is a constant is crucial in this paper.

[2] We recall that E is a finite ground set and $\mathcal{I} \subseteq 2^E$ is a nonempty family of subsets of E (*independent sets*) which have to satisfy the following two conditions: (i) $I \in \mathcal{I},\ J \subseteq I \ \Rightarrow\ J \in \mathcal{I}$ and (ii) $I, J \in \mathcal{I}, |I| > |J| \ \Rightarrow\ \exists z \in I \setminus J :\ J \cup \{z\} \in \mathcal{I}$. A *basis* is a maximal independent set. For all matroids used in this paper we make the usual assumption that independence of a set can be checked in polynomial time. For additional information on matroids, see e.g. [24].

problem is NP-complete. Due to space constraints, we omit the simple proof of the following theorem (which might be considered as part of folklore).

Theorem 1. *For $k \geq 2$, it is NP-complete to decide whether there is a feasible solution for k-*BUDGETED SHORTEST PATH*, k-*BUDGETED PERFECT MATCHING *and k-*BUDGETED SPANNING TREE *(and hence also for k-budgeted matroid basis and matroid intersection basis).*

The remaining problems in the above list have a common aspect: the set of solutions $\mathcal{F}$ forms an *independence system.* In other terms, for $S \in \mathcal{F}$ and $S' \subseteq S$, we have $S' \in \mathcal{F}$. For these problems, we present deterministic and randomized approximation schemes, based on a variety of techniques.

Our first result is a simple but powerful mechanism to transform a multi-criteria PTAS, i.e., a PTAS that might violate the budgets by a small multiplicative factor, into a *pure* PTAS, where no budget is violated. Similarly, a multi-criteria polynomial randomized-time approximation scheme (PRAS) can be transformed into a pure PRAS (see Section 2).

Theorem 2. (Feasibilization) *Let $\mathcal{P}_{ind}$ be a k-budgeted problem where the set of solutions $\mathcal{F}$ is an independence system. Suppose that we are given an algorithm $\mathcal{A}$ which, for any constant $\delta > 0$, computes in polynomial time an $(1-\delta)$ (resp., expected $(1-\delta)$) approximate solution to $\mathcal{P}_{ind}$ violating each budget by a factor at most $(1+\delta)$. Then there is a PTAS (resp., PRAS) for $\mathcal{P}_{ind}$.*

The basic idea is as follows. We show that a *good* solution exists even if we scale down the budgets by a small factor. This is done by applying a greedy discarding strategy similar to the greedy algorithm for KNAPSACK. Applying a multi-criteria PTAS (given as a black box) to the scaled problem gives a feasible solution for the original one, of weight *close* to the optimal weight.

To the best of our knowledge, this simple result was never observed before. Indeed, it implies improved approximation algorithms for a number of problems. A general construction by Papadimitriou and Yannakakis [18] provides multi-criteria PTASs (resp., PRASs) for problems whose exact version admits a pseudo-polynomial-time (PPT) deterministic (resp., Monte-Carlo) algorithm. We recall that the *exact version* of a given optimization problem asks for a feasible solution of exactly a given *target* weight. Combining their approach with our mechanism one obtains approximation schemes for several problems. For example, using the PPT-algorithm for EXACT FOREST in [2], one obtains a PTAS for k-BUDGETED FOREST. Similarly, the Monte-Carlo PPT-algorithm for EXACT MATCHING in [17] gives a PRAS for k-BUDGETED MATCHING. The Monte-Carlo PPT-algorithms for EXACT MATROID INTERSECTION INDEPENDENT SET in [5], which works in the special case of representable matroids[3], implies a PRAS for the corresponding budgeted problem.

Of course, one can also exploit multi-criteria approximation schemes obtained with different techniques. For example, exploiting the multi-criteria PTAS in

[3] A matroid $M = (E, \mathcal{I})$ is representable if its ground set E can be mapped in a bijective way to the columns of a matrix over some field, and $I \subseteq E$ is independent in M iff the corresponding columns are linearly independent.

[8] for k-BUDGETED MATCHING in bipartite graphs, which is based on iterative rounding, one obtains a PTAS for the same problem. Very recently [6], a multi-criteria PRAS for k-BUDGETED MATROID INDEPENDENT SET, based on dependent randomized rounding, has been presented. This implies a PRAS for k-BUDGETED MATROID INDEPENDENT SET.

Corollary 1. *There are PTASs for* k-BUDGETED FOREST *and* k-BUDGETED MATCHING *in bipartite graphs. There are PRASs for* k-BUDGETED MATCHING*,* k-BUDGETED MATROID INDEPENDENT SET*, and* k-BUDGETED MATROID INTERSECTION *in representable matroids.*

Based on a different, more direct approach, we are able to turn the PRAS for k-BUDGETED MATROID INDEPENDENT SET into a PTAS. The main insight is the following structural property of faces of the matroid polytope which might be of independent interest (proof in Section 3).

Theorem 3. *Let $M = (E, \mathcal{I})$ be a matroid and let F be a face of dimension d of the matroid polytope*[4] *$P_{\mathcal{I}}$. Then any $x \in F$ has at most $2d$ non-integral components. Furthermore, the sum of all fractional components of x is at most d.*

A PTAS can then be easily derived as follows. We first guess the k/ε elements E_H of largest weight in the optimum solution in a preliminary phase, and reduce the problem consequently. This guessing step guarantees that the maximum weight $\mathrm{w}_{\max}$ of an element in the reduced problem satisfies $k\mathrm{w}_{\max} \leq \varepsilon \mathrm{w}(E_H)$. For the reduced problem, we compute an optimal fractional vertex solution x^* to the LP which seeks to find a maximum weight point in the matroid polytope intersected with the k budget constraints. Since x^* is chosen to be a vertex solution, and only k linear constraints are added to the matroid polytope, x^* lies on a face of the matroid polytope of dimension at most k. We then round down the fractional components of x^* to obtain an incidence vector $\overline{x}$ which corresponds to some independent set E_L. By Theorem 3, $|x^* - \overline{x}| \leq k$, and hence, $\mathrm{w}(E_L) \geq \mathrm{w}(x^*) - k\mathrm{w}_{\max}$. Then, it is not hard to see that $E_H \cup E_L$ is a $(1-\varepsilon)$-approximate feasible solution for the starting problem.

Corollary 2. *There is a PTAS for* k-BUDGETED MATROID INDEPENDENT SET.

Eventually, we present a PTAS (rather than a PRAS as in Corollary 1) for 2-BUDGETED MATCHING (see Section 4).

Theorem 4. *There is a PTAS for* 2-BUDGETED MATCHING.

Our PTAS works as follows. Let us confuse a matching M with the associated incidence vector x_M. We initially compute an optimal fractional matching x^*, and express it as the convex combination $x^* = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3$ of three matchings x_1, x_2, and x_3. Then we exploit a *patching procedure* which, given two matchings x' and x'' with high Lagrangian weight and a parameter $\mu \in [0, 1]$, computes a matching z which is not longer than $x_\mu := \mu x' + (1-\mu)x''$ with

[4] For some given matroid $M = (E, \mathcal{I})$, the corresponding matroid polytope $P_{\mathcal{I}}$ is the convex hull of the incidence vectors of all independent sets.

respect to both lengths, and has a comparable weight. This procedure is applied twice: first on the matchings x_1 and x_2 with parameter $\mu = \alpha_1/(\alpha_1 + \alpha_2)$, hence getting a matching z'. Second, on the two matchings z' and x_3 with parameter $\mu = (\alpha_1 + \alpha_2)/(\alpha_1 + \alpha_2 + \alpha_3)$. The resulting matching z'' is feasible and almost optimal (modulo a preliminary guessing step).

Our patching procedure relies on a topological property of curves in $\mathbb{R}^2$, that we prove via Jordan's curve theorem [15]. An extension of the property above to curves in $\mathbb{R}^k$ would imply a PTAS for k-BUDGETED MATCHING: this is left as an interesting open problem (details are omitted for lack of space).

1.2 Related Work

There are a few general tools for designing approximation algorithms for budgeted problems. One basic approach is combining *dynamic programming* (which solves the problem for polynomial weights and lengths) with *rounding and scaling* techniques (to reduce the problem to the case of polynomial quantities). This leads for example to the FPTAS for 1-BUDGETED SHORTEST PATH [10,14,25]. Another fundamental technique is the *Lagrangian relaxation method.* The basic idea is relaxing the budget constraints, and lifting them into the objective function, where they are weighted by Lagrangian multipliers. Solving the relaxed problem, one obtains two or more solutions with optimal Lagrangian weight, which can - if needed - be patched together to get a good solution for the original problem. Demonstrating this method, Goemans and Ravi [21] gave a PTAS for 1-BUDGETED SPANNING TREE, which also extends to 1-BUDGETED MATROID BASIS. Using the same approach, with an involved patching step, Berger, Bonifaci, Grandoni, and Schäfer [3] obtained a PTAS for 1-BUDGETED MATCHING and 1-BUDGETED MATROID INTERSECTION INDEPENDENT SET. Their approach does not seem to generalize to the case of multiple budget constraints.

The techniques above apply to the case of one budget. Not much is known for problems with two or more budgets. However, often *multi-criteria* approximation schemes are known, which provide a $(1 - \varepsilon)$-approximate solution violating the budgets by a factor $(1 + \varepsilon)$. First of all, there is a very general technique by Papadimitriou and Yannakakis [18], based on the construction of ε-approximate Pareto curves. Given an optimization problem with multiple objectives, the *Pareto curve* consists of the set of solutions S such that there is no solution S' which is strictly better than S (in a vectorial sense). Papadimitriou and Yannakakis show that, for any constant $\varepsilon > 0$, there always exists a polynomial-size ε-approximate Pareto curve $\mathcal{A}$, i.e., a set of solutions such that every solution in the Pareto curve is within a factor of $(1+\varepsilon)$ from some solution in $\mathcal{A}$ on each objective. Furthermore, this approximate curve can be constructed in polynomial time in the size of the input and $1/\varepsilon$ whenever there exists a PPT algorithm for the associated exact problem. This implies multi-criteria FPTASs for k-BUDGETED SPANNING TREE and k-BUDGETED SHORTEST PATH. Furthermore, it implies a multi-criteria FPRAS for k-BUDGETED (PERFECT) MATCHING. The latter result exploits the Monte-Carlo PPT algorithm for EXACT MATCHING in [17]. Our PRAS improves on these results, approximation-wise (the running time is larger in our case).

Recently, Grandoni, Ravi and Singh [8] showed that the *iterative rounding* technique is an alternative way to achieve similar (or better) results. Using this method they obtain a multi-criteria PTAS for k-BUDGETED SPANNING TREE, which computes a solution of optimal cost violating each budget by a factor $(1+\varepsilon)$. This improves, approximation-wise, on the result in [18] for the same problem (where the solution returned is suboptimal). The authors also show how to obtain a deterministic (rather than randomized [18]) multi-criteria PTAS for k-BUDGETED MATCHING in bipartite graphs.

All mentioned problems are easy in the unbudgeted version. Given an NP-hard unbudgeted problem which admits a ρ approximation, the *parametric search* technique in [16] provides a multi-criteria $k\rho$ approximation algorithm violating each budget by a factor $k\rho$ for the corresponding problem with k budgets. Other techniques lead to logarithmic approximation factors (see, e.g., [4,19,20]).

2 A Feasibilization Mechanism

Proof (Theorem 2). Let $\varepsilon \in (0,1]$ be a given constant, with $1/\varepsilon \in \mathbb{N}$. Consider the following algorithm. Initially we guess the $h = k/\varepsilon$ elements E_H of OPT of largest weight, and reduce the problem consequently[5], hence getting a problem $\mathcal{P}'$. Then we scale down all the budgets by a factor $(1-\delta)$, and solve the resulting problem $\mathcal{P}''$ by means of $\mathcal{A}$, where $\delta = \varepsilon/(k+1)$. Let E_L be the solution returned by $\mathcal{A}$. We finally output $E_H \cup E_L$.

Let OPT' and OPT'' be the optimum solution to problems $\mathcal{P}'$ and $\mathcal{P}''$, respectively. We also denote by L'_i and L''_i the ith budget in the two problems, respectively. Let w_{max} be the largest weight in $\mathcal{P}'$ and $\mathcal{P}''$. We observe that trivially: (a) $\mathrm{w}(OPT) = \mathrm{w}(E_H) + \mathrm{w}(OPT')$ and (b) $\mathrm{w}_{max} \le \mathrm{w}(E_H)/h$.

Let us show that (c) $\mathrm{w}(OPT'') \ge \mathrm{w}(OPT')(1-k\delta) - k\mathrm{w}_{max}$. Consider the following process: for each length function i, we remove from OPT' the element e with smallest ratio $\mathrm{w}(e)/\ell_i(e)$ until $\ell_i(OPT') \le (1-\delta)\mathrm{L}'_i$. Let E_i be the set of elements removed. It is not hard to see that $\mathrm{w}(E_i) \le \delta\mathrm{w}(OPT') + \mathrm{w}_{max}$. It follows that $OPT' - \cup_i E_i$ is a feasible solution for $\mathcal{P}''$ of weight at least $w(OPT')(1-\delta k) - k\mathrm{w}_{max}$, proving (c).

We observe that E_L is feasible for $\mathcal{P}'$ since, for each i, $\ell_i(E_L) \le (1+\delta)L''_i = (1+\delta)(1-\delta)L'_i \le L'_i$. As a consequence, the returned solution $E_H \cup E_L$ is feasible. Moreover, when $\mathcal{A}$ is deterministic, we have

$$\begin{aligned} w(E_H) + w(E_L) &\ge w(E_H) + (1-\delta)w(OPT'') \\ &\overset{(c)}{\ge} w(E_H) + (1-\delta)(w(OPT')(1-\delta k) - k\mathrm{w}_{max}) \end{aligned}$$

[5] As usual, by reducing we mean decreasing each budget L_i by $\ell_i(E_H)$ and removing all the elements of weight strictly larger than $\min_{e\in E_H} \mathrm{w}(e)$. By guessing we mean trying all the $O(m^h)$ subsets of h elements.

$$\begin{aligned}&\overset{(b)}{\geq} (1-k/h)w(E_H) + (1-\delta(k+1))w(OPT')\\&\geq (1-\varepsilon)(w(E_H)+w(OPT')) \overset{(a)}{=} (1-\varepsilon)w(OPT).\end{aligned}$$

The same bound holds in expectation when $\mathcal{A}$ is randomized.

3 A PTAS for k-Budgeted Matroid Independent Set

It is convenient to consider weights w and lengths ℓ_i as vectors in $\mathbb{Q}^E$. We denote by ℓ the matrix whose ith column is ℓ_i, and let $L = (L_1, \dots, L_k)^T$. A *rank function* $r : 2^E \to \mathbb{N}$ is associated to every matroid $M = (E, \mathcal{I})$; it is defined by $r(S) = \max\{|J| \mid J \subseteq S, J \in \mathcal{I}\}$. The *matroid polytope* $P_{\mathcal{I}}$ is the convex hull of the characteristic vectors χ_I of the independent sets $I \in \mathcal{I}$ and is described by the following set of inequalities: $P_{\mathcal{I}} = \text{conv}\{\chi_I : I \in \mathcal{I}\} = \{x \geq 0 : x(S) \leq r(S)\ \forall S \subseteq E\}$. As usual, $x(S) := \sum_{e \in S} x(e)$[6].

Proof (Theorem 3). Let $m = |E|$. We assume that the matroid polytope has full dimension, i.e., $\dim(P_{\mathcal{I}}) = m$, or equivalently, every element $e \in E$ is independent. This can be assumed wlog since if $\{e\} \notin \mathcal{I}$ for some $e \in E$, then we can reduce the matroid by deleting element e. Since $\dim(P_{\mathcal{I}}) = m$ and $\dim(F) = d$, F can be described by the inequality system of $P_{\mathcal{I}}$, where $m-d$ linearly independent inequalities used in the description of $P_{\mathcal{I}}$ are turned into equalities. More precisely, there are $N \subseteq E$ and $A_1, \dots, A_k \subseteq E$ such that

$$F = \{x \in P_{\mathcal{I}} \mid x(e) = 0\ \forall e \in N, x(A_i) = r(A_i)\ \forall i \in \{1, \dots, k\}\},$$

and $|N| + k = m - d$. By standard uncrossing arguments, we can assume that the sets A_i form a chain, i.e., $A_1 \subsetneq A_2 \subsetneq \cdots \subsetneq A_k$ (see for example [9,12] for further information on combinatorial uncrossing). We prove the claim by induction on the number of elements of the matroid. The theorem clearly holds for matroids with a ground set of cardinality one. First assume $N \neq \emptyset$ and let $e \in N$. Let M' be the matroid obtained from M by deleting e, and let F' be the projection of F onto the coordinates corresponding to $N \setminus \{e\}$. Since F' is a face of M', the claim follows by induction. Henceforth, we assume $N = \emptyset$ which implies $k = m - d$. Let $A_0 = \emptyset$ and $B_i = A_i \setminus A_{i-1}$ for $i \in \{1, \dots, k\}$. In the following we show that we can assume

$$0 < r(A_i) - r(A_{i-1}) < |B_i| \quad \forall\, i \in \{1, \dots, k\}. \tag{1}$$

Notice that $0 \leq r(A_i) - r(A_{i-1}) \leq |B_i|$ clearly holds by standard properties of rank functions (see [24] for more details). Assume that there is $i \in \{1, \dots, k\}$ with $r(A_i) = r(A_{i-1})$. Since all points $x \in F$ satisfy $x(A_i) = r(A_i)$ and $x(A_{i-1}) = r(A_{i-1})$, we have $x(B_i) = 0$. Hence for any $e \in B_i$, we have $x(e) = 0$ for $x \in F$. Again, we can delete e from the matroid, hence obtaining a smaller matroid for which the claim holds by the inductive hypothesis. Therefore, we can assume $r(A_i) > r(A_{i-1})$ which implies the left inequality in (1).

[6] See [24] for more details and omitted standard definitions.

For the right inequality assume that there is $i \in \{1, \dots, k\}$ with $r(A_i) - r(A_{i-1}) = |B_i|$. Hence, every $x \in F$ satisfies $x(B_i) = |B_i|$, implying $x(e) = 1$ for all $e \in B_i$. Let $e \in B_i$, and let F' be the projection of the face F onto the components $N \setminus \{e\}$. Since F' is a face of the matroid M' obtained from M by contracting e, the result follows again by the inductive hypothesis.

Henceforth, we assume that (1) holds. This implies in particular that $|B_i| > 1$ for $i \in \{1, \dots, k\}$. Since $\sum_{i=1}^{k} |B_i| \leq m$, we have $k \leq m/2$, which together with $k = m - d$ implies $d \geq m/2$. The claim of the theorem that $x \in F$ has at most $2d$ non-integral components is thus trivial in this case.

To prove the second part of the theorem we show that if (1) holds then $x(E) \leq d$ for $x \in F$. For $x \in F$ we have

$$\begin{aligned} x(E) &= x(E \setminus A_k) + \textstyle\sum_{i=1}^{k} x(B_i) \leq |E| - |A_k| + \sum_{i=1}^{k}(r(A_i) - r(A_{i-1})) \\ &\leq |E| - |A_k| + \textstyle\sum_{i=1}^{k}(|A_i| - |A_{i-1}| - 1) = m - k = d, \end{aligned}$$

where the first inequality follows from $x(E \setminus A_k) \leq |E \setminus A_k|$ and $x(B_i) = r(A_i) - r(A_{i-1})$, and the second inequality follows from (1).

4 A PTAS for 2-BUDGETED MATCHING

In this section we present our PTAS for 2-BUDGETED MATCHING. We denote by $\mathcal{M}$ the set of incidence vectors of matchings. With a slight abuse of terminology we call the elements in $\mathcal{M}$ matchings. Let $P_{\mathcal{M}}$ be the matching polyhedron. Analogously to Section 3, let $\ell = (\ell_1, \ell_2)$ and $\mathrm{L} = (\mathrm{L}_1, \mathrm{L}_2)^T$. A feasible solution in this framework is a matching $x \in \mathcal{M}$ such that $\ell^T x \leq \mathrm{L}$. For two elements $z', z'' \in [0,1]^E$, we define their *symmetric difference* $z' \Delta z'' \in [0,1]^E$ by $(z' \Delta z'')(e) = |z'(e) - z''(e)|$ for all $e \in E$. In particular, if z' and z'' are incidence vectors, then their symmetric difference as defined above corresponds indeed to the symmetric difference in the usual sense. Recall that, when z' and z'' are matchings, $z' \Delta z''$ consists of a set of node-disjoint paths and cycles.

We start by presenting a property of curves in $\mathbb{R}^2$. This property is used to derive the mentioned *patching procedure*. Eventually, we describe and analyze our PTAS.

A Property of Curves in $\mathbb{R}^2$. We next describe a topological property of polygonal curves in $\mathbb{R}^2$, which will be crucial in our proof[7]. A *curve* in $\mathbb{R}^2$ is a continuous function $f : [0, \tau] \to \mathbb{R}^2$ for some $\tau \in \mathbb{R}_+$. A curve is called *polygonal* if it is piecewise linear. For $a \in [0, \tau]$, let $f^a : [0, \tau] \to \mathbb{R}^2$ be the following curve.

$$f^a(t) = \begin{cases} f(t+a) - f(a) + f(0) & \text{if } t + a < \tau, \\ f(\tau) - f(a) + f(a + t - \tau) & \text{if } t + a \geq \tau. \end{cases}$$

[7] The lemma even holds for general (non-polygonal) curves. However, since we only need polygonal curves in our setting we restrict ourselves to this case since it simplifies the exposition.

Observe that $f^a(0) = f(0)$ and $f^a(\tau) = f(\tau)$ for any $a \in [0,\tau]$. The next lemma shows that any point x on the segment between $f(0)$ and $f(\tau)$ is contained in some curve f^a.

Lemma 1. *Let $f : [0,\tau] \to \mathbb{R}^2$ be a polygonal curve, and let $\mu \in [0,1]$. Then there are $a, t \in [0,\tau]$ such that $f^a(t) = \mu f(0) + (1-\mu) f(\tau)$.*

We next give an intuitive description of the proof of the lemma: a formal proof is postponed to the journal version of the paper. Let $f = (f_1, f_2)$. Since the statement of the lemma is independent of changes in the coordinate system (and the claim is trivial for $f(0) = f(\tau)$), we can assume that $f(0) = (0,0)$ and $f(\tau) = (r,0)$ for some $r > 0$. The Gasoline Lemma [3] states that there is $a_1 \in [0,\tau]$ such that $f_2^{a_1}(t) \geq 0\ \forall t \in [0,\tau]$. In particular, this condition is satisfied by choosing $a_1 \in \arg\min\{f_2(t) \mid t \in [0,\tau]\}$. Analogously, for $a_2 \in \arg\max\{f_2(t) \mid t \in [0,\tau]\}$, $f_2^{a_2}(t) \leq 0\ \forall t \in [0,\tau]$. Hence, we have two curves, f^{a_1} and f^{a_2}, one above and the other below the x-axis, both with the same endpoints $(0,0)$ and $(r,0)$. Furthermore, for a ranging from a_1 to a_2 (in a circular sense), the curve f^a continuously transforms from f^{a_1} to f^{a_2}, always maintaining the same endpoints. Then it is intuitively clear that the union of the curves f^a spans all the points on the segment from $(0,0)$ to $(r,0)$, hence proving the claim.

The Patching Procedure. In this section we describe a *patching procedure* which, given two matchings x' and x'' and a parameter $\mu \in [0,1]$, computes a matching z satisfying $\ell^T z \leq \ell^T x_\mu$, where $x_\mu := \mu x' + (1-\mu)x''$ is a convex combination of the first two matchings. Furthermore, the weight $\mathrm{w}^T z$ is *close* to $\mathrm{w}^T x_\mu$, provided that x' and x'' have a *sufficiently* large Lagrangian weight, which is defined as follows. Let $\lambda_1^*, \lambda_2^* \in \mathbb{R}_+$ be a pair of optimal dual multipliers for the budgets in the linear program $\max\{\mathrm{w}^T x \mid x \in P_{\mathcal{M}}, \ell^T x \leq L\}$. The *Langrangian weight* of $x \in [0,1]^E$ is $\mathcal{L}(x) = \mathrm{w}^T x - (\lambda_1^*, \lambda_2^*)(\ell^T x - \mathrm{L})$. Notice, that by the theory of Lagrangian duality we have $\mathrm{w}^* = \max\{\mathcal{L}(x) \mid x \in P_{\mathcal{M}}\}$, where w^* is the weight of an optimal LP solution, i.e., $\mathrm{w}^* = \max\{\mathrm{w}^T x \mid x \in P_{\mathcal{M}}, \ell^T x \leq L\}$ (see [13] for more information on Lagrangian duality).

We need the following notion of almost matching.

Definition 1. *For $r \in \mathbb{N}$, an* r-almost matching *in G is a (possibly fractional) vector $y \in [0,1]^E$ such that it is possible to set at most r components of y to zero to obtain a matching.*

We denote by $\mathcal{M}_r$ the set of all r-almost matchings in G. Given an r-almost matching y, we let a *corresponding matching* $z \in \mathcal{M}$ be a matching obtained by setting to zero the fractional components of y, and then computing a maximal matching in the resulting set of edges (in particular, we might need to set to 0 some 1 entries of y to obtain z). Notice that $\mathrm{w}^T z \geq \mathrm{w}^T y - r\mathrm{w}_{\max}$, where $w_{\max}$ is the largest weight.

Our patching procedure first constructs a 2-almost matching y, and then returns a corresponding matching z. We next show how to compute y. Let us restrict our attention to the following set of candidate 2-almost matchings. Recall that $s = x' \Delta x''$ is a set of paths and cycles. We construct an auxiliary graph

C, consisting of one cycle $(e_0, e_1, \ldots, e_{\tau-1})$, with the following property: there is a bijective mapping between the edges of C and the edges of s such that two consecutive edges of C are either consecutive in some path/cycle or belong to different paths/cycles. This can be easily achieved by cutting each cycle, appending the resulting set of paths one to the other, and gluing together the endpoints of the obtained path. For $t \in [0, \tau]$, we define $s(t) \in [0,1]^E$ as

$$(s(t))(e) = \begin{cases} 1 & \text{if } e = e_i,\ i < \lfloor t \rfloor; \\ t - \lfloor t \rfloor & \text{if } e = e_i,\ i = \lfloor t \rfloor; \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, for $a, t \in [0, \tau]$, we define

$$[0,1]^E \ni s^a(t) = \begin{cases} s(a+t) - s(a) & \text{if } a + t \leq \tau; \\ s(a+t-\tau) + s(\tau) - s(a) & \text{if } a + t > \tau. \end{cases}$$

Intuitively, a and $(a+t) \pmod{\tau}$ define a (fractional) subpath of C, and $s^a(t)$ is the (fractional) incidence vector corresponding to that subpath. Additionally we define $y^a(t) := x' \triangle s^a(t)$. Note that $y^a(t)$ is equal to x' and x'' for $t = 0$ and $t = \tau$, respectively.

Lemma 2. *For any $a, t \in [0, \tau]$, $y^a(t)$ is a 2-almost matching.*

Proof. One can easily observe that a matching can be obtained by setting the two components of $y^a(t)$ to zero that correspond to the edges $e_{\lfloor a \rfloor}$ and $e_{\lfloor (a+t) \pmod{\tau} \rfloor}$.

The following lemma shows that, in polynomial time, one can find a 2-almost matching y with lengths $\ell^T y$ equal to the lengths of any convex combination of the two matchings x' and x''.

Lemma 3. *Let $\mu \in [0,1]$ and $x_\mu = \mu x' + (1-\mu)x''$. In polynomial time, $a, t \in [0, \tau]$ can be determined such that $\ell^T y^a(t) = \ell^T x_\mu$.*

Proof. Let $f : [0, \tau] \to \mathbb{R}^2$ be the polygonal curve defined by $f(t) = \ell^T y^0(t)$. Since $f(0) = \ell^T x'$ and $f(\tau) = \ell^T x''$, we have by Lemma 1 that there exists $a, t \in [0, \tau]$ such that $f^a(t) = \ell^T x_\mu$. Since $f^a(t) = \ell^T y^a(t)$, $y := y^a(t)$ satisfies the claim. The values of $\lfloor a \rfloor$ and $\lfloor a + t \rfloor$ can be guessed in polynomial time by considering $O(n^2)$ possibilities. Given those two rounded values, the actual values of a and t can be obtained by solving a linear program with a constant number of variables and constraints.

Our patching procedure computes a 2-almost matching $y = y^a(t)$ with $\ell^T y = \ell^T x_\mu$, exploiting the lemma above, and then returns a corresponding matching z, by applying the procedure explained in the proof of Lemma 2. Trivially, $\ell^T z \leq \ell^T y = \ell^T x_\mu$. We next show that, if x' and x'' have sufficiently large Lagrangian weight, then the weight of z is *close* to the weight of x_μ.

Lemma 4. *Assume $\mathcal{L}(x') \geq \mathrm{w}^* - \Gamma$ and $\mathcal{L}(x'') \geq \mathrm{w}^* - \Gamma$ for some $\Gamma \in \mathbb{R}_+$. Then the matching z returned by the patching procedure satisfies $\mathrm{w}^T z \geq \mathrm{w}^T x_\mu - 2\mathrm{w}_{\max} - \Gamma$ and $\ell^T z \leq \ell^T x_\mu$.*

Proof. By Lemma 3 we have $\ell^T y = \ell^T x_\mu$, and since $z \leq y$, we get $\ell^T z \leq \ell^T x_\mu$. Let $\overline{x}_\mu = x' + x'' - x_\mu = (1-\mu)x' + \mu x''$. Since $\mathcal{L}(x') \geq \mathrm{w}^* - \Gamma$, $\mathcal{L}(x'') \geq \mathrm{w}^* - \Gamma$ and $\mathcal{L}$ is linear, we have $\mathcal{L}(x_\mu) \geq \mathrm{w}^* - \Gamma$ and $\mathcal{L}(\overline{x}_\mu) \geq \mathrm{w}^* - \Gamma$. Recall that $y = y^a(t)$ for a proper choice of $a, t \in [0, \tau]$. Let $\overline{y} := x' + x'' - y$. Notice that $\overline{y} = y^{a'}(\tau - t)$ where $a' = (a + t) \pmod{\tau}$, and hence, $\overline{y}$ is also a 2-almost matching by Lemma 2. Let $\overline{z}$ be the matching corresponding to $\overline{y}$ obtained by applying the procedure explained in the proof of Lemma 2 to $\overline{y}$. Notice that the pairs (z, y) and $(\overline{z}, \overline{y})$ differ on the same two (or less) components. Hence

$$\mathrm{w}^T z + \mathrm{w}^T \overline{z} + 2\mathrm{w}_{\max} \geq \mathrm{w}^T y + \mathrm{w}^T \overline{y} = \mathrm{w}^T x_\mu + \mathrm{w}^T \overline{x}_\mu. \tag{2}$$

Since $y + \overline{y} = x_\mu + \overline{x}_\mu$ and $\ell^T y = \ell^T x_\mu$, we get $\ell^T \overline{y} = \ell^T \overline{x}_\mu$. Thus, $\ell^T \overline{z} \leq \ell^T \overline{x}_\mu$ since $\overline{z} \leq \overline{y}$. This can be rewritten as $\mathcal{L}(\overline{z}) - \mathrm{w}^T \overline{z} \geq \mathcal{L}(\overline{x}_\mu) - \mathrm{w}^T \overline{x}_\mu$. Since $\mathcal{L}(\overline{x}_\mu) \geq \mathrm{w}^* - \Gamma$ and $\mathcal{L}(\overline{z}) \leq \mathrm{w}^*$, we obtain $\mathrm{w}^T \overline{z} \leq \mathrm{w}^T \overline{x}_\mu + \Gamma$. Combining this result with (2) implies $\mathrm{w}^T z \geq \mathrm{w}^T x_\mu - 2\mathrm{w}_{\max} - \Gamma$.

The Algorithm. Our PTAS works as follows. First it guesses the $6/\varepsilon$ heaviest edges E_H in the optimum solution, and reduces the problem consequently. Then it computes a vertex $x^* \in P_\mathcal{M}$ of the polytope $\{x \in P_\mathcal{M} \mid \ell^T x \leq L\}$ of maximum weight $\mathrm{w}^* := \mathrm{w}^T x^*$. As x^* is a vertex solution of the polytope $P_\mathcal{M}$ with two additional constraints, it lies on a face of $P_\mathcal{M}$ of dimension at most two. Hence, by Carathéodory's Theorem, x^* can be expressed as a convex combination $x^* = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3$ of three matchings $x_1, x_2, x_3 \in P_\mathcal{M}$. Let $\mu' = \alpha_1/(\alpha_1 + \alpha_2)$ and $\mu'' = (\alpha_1 + \alpha_2)/(\alpha_1 + \alpha_2 + \alpha_3)$. Applying Lemma 4 to x_1 and x_2 with $\mu = \mu'$, a matching z' is obtained. Applying Lemma 4 to z' and x_3 with $\mu = \mu''$, we obtain a matching z''. The algorithm returns z'' and E_H.

Proof (Theorem 4). Consider the algorithm above. The initial guessing can be performed in $O(|E|^{6/\varepsilon})$ time. Since it is possible to efficiently separate over $P_\mathcal{M}$, x^* can be computed in polynomial time [24]. The same holds for the decomposition of x^* into three matchings by standard techniques (see for example [23]). Lemma 3 implies that the patching can be done in polynomial time. Hence the proposed algorithm runs in polynomial time as claimed.

Since $\mathcal{L}(x^*) = \mathrm{w}^*$ and $\mathcal{L}(x) \leq \mathrm{w}^*$ for $x \in P_\mathcal{M}$, we get $\mathcal{L}(x_1) = \mathcal{L}(x_2) = \mathcal{L}(x_3) = \mathrm{w}^*$. Let $u := \mu' x_1 + (1-\mu')x_2$ and $v := \mu'' z' + (1-\mu'')x_3$. By Lemma 4, matching z' satisfies $\ell^T z' \leq \ell^T u$ and $\mathrm{w}^T z' \geq \mathrm{w}^T u - 2\mathrm{w}_{\max}$. Since u is a convex combination of x_1 and x_2, we have $\mathcal{L}(u) = \mathrm{w}^*$. Furthermore, by the relations between the lengths and weight of z' and u, we get $\mathcal{L}(z') \geq \mathcal{L}(u) - 2\mathrm{w}_{\max} = \mathrm{w}^* - 2\mathrm{w}_{\max}$.

By Lemma 4, matching z'' satisfies $\ell^T z'' \leq \ell^T v$ and $\mathrm{w}^T z'' \geq \mathrm{w}^T v - 4\mathrm{w}_{\max}$. We observe that z'' satisfies the budget constraints since

$$\ell^T z'' \leq \ell^T v = \ell^T((\alpha_1 + \alpha_2)z' + \alpha_3 x_3) \leq \ell^T((\alpha_1 + \alpha_2)u + \alpha_3 x_3) = \ell^T x^* \leq \mathrm{L}.$$

Furthermore,

$$\begin{aligned} \mathrm{w}^T z'' &\geq \mathrm{w}^T v - 4\mathrm{w}_{\max} = \mathrm{w}^T((\alpha_1 + \alpha_2)z' + \alpha_3 x_3) - 4\mathrm{w}_{\max} \\ &\geq \mathrm{w}^T((\alpha_1 + \alpha_2)u + \alpha_3 x_3) - 6\mathrm{w}_{\max} = \mathrm{w}^* - 6\mathrm{w}_{\max}. \end{aligned}$$

Let OPT' be an optimum solution to the reduced problem. Of course, $\mathrm{w}^* \geq \mathrm{w}(OPT')$. Furthermore, the weight of the guessed edges E_H is at least $6/\varepsilon\, w_{max}$. Since $\mathrm{w}(OPT) = \mathrm{w}(E_H) + \mathrm{w}(OPT')$, we can conclude that the solution returned by the algorithm has weight at least $\mathrm{w}(E_H)(1-\varepsilon) + \mathrm{w}(OPT') \geq (1-\varepsilon)\mathrm{w}(OPT)$.

References

1. Aggarwal, V., Aneja, Y.P., Nair, K.P.K.: Minimal spanning tree subject to a side constraint. Computers & Operations Research 9, 287–296 (1982)
2. Barahona, F., Pulleyblank, W.R.: Exact arborescences, matchings and cycles. Discrete Applied Mathematics 16(2), 91–99 (1987)
3. Berger, A., Bonifaci, V., Grandoni, F., Schäfer, G.: Budgeted matching and budgeted matroid intersection via the gasoline puzzle. To appear in Mathematical Programming, Preliminary version in IPCO 2008
4. Bilu, V., Goyal, V., Ravi, R., Singh, M.: On the crossing spanning tree problem. In: Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds.) RANDOM 2004 and APPROX 2004. LNCS, vol. 3122, pp. 51–64. Springer, Heidelberg (2004)
5. Camerini, P., Galbiati, G., Maffioli, F.: Random pseudo-polynomial algorithms for exact matroid problems. Journal of Algorithms 13, 258–273 (1992)
6. Chekuri, C., Vondrák, J., Zenklusen, R.: Dependent randomized rounding for matroid polytopes and applications (2009), `http://arxiv.org/abs/0909.4348`
7. Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness. W.H. Freeman, New York (1979)
8. Grandoni, F., Ravi, R., Singh, M.: Iterative rounding for multi-objective optimization problems. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 95–106. Springer, Heidelberg (2009)
9. Hurkens, C.A., Lovász, L., Schrijver, A., Tardos, E.: How to tidy up your set system. Combinatorics, 309–314 (1988)
10. Hassin, R.: Approximation schemes for the restricted shortest path problem. Mathematics of Operation Research 17(1), 36–42 (1992)
11. Hassin, R., Levin, A.: An efficient polynomial time approximation scheme for the constrained minimum spanning tree problem using matroid intersection. SIAM Journal on Computing 33(2), 261–268 (2004)
12. Jain, K.: A factor 2 approximation algorithm for the generalized Steiner network problem. Combinatorica 21, 39–60 (2001)
13. Korte, B., Vygen, J.: Combinatorial optimization. Springer, Heidelberg (2008)
14. Lorenz, D., Raz, D.: A simple efficient approximation scheme for the restricted shortest paths problem. Operations Research Letters 28, 213–219 (2001)
15. Munkres, J.R.: Topology, 2nd edn. Prentice-Hall, Englewood Cliffs (2000)
16. Marathe, M.V., Ravi, R., Sundaram, R., Ravi, S.S., Rosenkrantz, D.J., Hunt III, H.B.: Bicriteria network design problems. In: Fülöp, Z., Gecseg, F. (eds.) ICALP 1995. LNCS, vol. 944, pp. 487–498. Springer, Heidelberg (1995)
17. Mulmuley, K., Vazirani, U., Vazirani, V.: Matching is as easy as matrix inversion. Combinatorica 7(1), 101–104 (1987)
18. Papadimitriou, C.H., Yannakakis, M.: On the approximability of trade-offs and optimal access of Web sources. In: FOCS, pp. 86–92 (2000)
19. Ravi, R.: Rapid rumor ramification: Approximating the minimum broadcast time. In: FOCS, pp. 202–213 (1994)

20. Ravi, R.: Matching based augmentations for approximating connectivity problems. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 13–24. Springer, Heidelberg (2006) (invited lecture)
21. Ravi, R., Goemans, M.X.: The constrained minimum spanning tree problem (extended abstract). In: Karlsson, R., Lingas, A. (eds.) SWAT 1996. LNCS, vol. 1097, pp. 66–75. Springer, Heidelberg (1996)
22. Ravi, R., Marathe, M.V., Ravi, S.S., Rosenkrantz, D.J., Hunt, H.B.: Many birds with one stone: Multi-objective approximation algorithms. In: STOC, pp. 438–447 (1993)
23. Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons, Chichester (1998)
24. Schrijver, A.: Combinatorial optimization, polyhedra and efficiency. Springer, Heidelberg (2003)
25. Warburton, A.: Approximation of Pareto optima in multiple-objective, shortest path problems. Operations Research 35, 70–79 (1987)

Algorithmic Meta-theorems for Restrictions of Treewidth

Michael Lampis

Computer Science Department,
Graduate Center, City University of New York
mlampis@gc.cuny.edu

Abstract. Possibly the most famous algorithmic meta-theorem is Courcelle's theorem, which states that all MSO-expressible graph properties are decidable in linear time for graphs of bounded treewidth. Unfortunately, the running time's dependence on the formula describing the problem is in general a tower of exponentials of unbounded height, and there exist lower bounds proving that this cannot be improved even if we restrict ourselves to deciding FO logic on trees.

We investigate whether this parameter dependence can be improved by focusing on two proper subclasses of the class of bounded treewidth graphs: graphs of bounded vertex cover and graphs of bounded max-leaf number. We prove stronger algorithmic meta-theorems for these more restricted classes of graphs. More specifically, we show it is possible to decide any FO property in both of these classes with a singly exponential parameter dependence and that it is possible to decide MSO logic on graphs of bounded vertex cover with a doubly exponential parameter dependence. We also prove lower bound results which show that our upper bounds cannot be improved significantly, under widely believed complexity assumptions. Our work addresses an open problem posed by Michael Fellows.

1 Introduction

Algorithmic metatheorems are general statements of the form "*All problems sharing property P, restricted to a class of inputs I can be solved efficiently*". The archetypal, and possibly most celebrated, such metatheorem is Courcelle's theorem which states that every graph property expressible in monadic second-order (MSO_2) logic is decidable in linear time if restricted to graphs of bounded treewidth [3]. Metatheorems have been a subject of intensive research in the last years producing a wealth of interesting results. Some representative examples of metatheorems with a flavor similar to Courcelle's can be found in the work of Frick and Grohe [12], where it is shown that all properties expressible in first order (FO) logic are solvable in linear time on planar graphs, and the work of Dawar et al. [5], where it is shown that all FO-definable optimisation problems admit a PTAS on graphs excluding a fixed minor (see [14] and [15] for more results on the topic).

Many interesting extensions have followed Courcelle's seminal result: for instance, Courcelle's theorem has been extended to logics more suitable for the expression of optimisation problems [1]. It has also been investigated whether it's possible to obtain similar results for larger graph classes (see [4] for a metatheorem for bounded cliquewidth graphs, [11] for corresponding hardness results and [17] for hardness results for graphs of small but unbounded treewidth). Finally, lower bound results have been shown proving that the running times predicted by Courcelle's theorem can not be improved significantly in general [13].

This lower bound result is one of the main motivations of this work, because in some ways it is quite devastating. Though Courcelle's theorem shows that a vast class of problems is solvable in linear time on graphs of bounded treewidth, the "hidden constant" in this running time, that is, the running time's dependence on the input's other parameters, which are the graph's treewidth and the formula describing the problem, is in fact (in the worst case) a tower of exponentials. Unfortunately, in [13] it is shown that this tower of exponentials is unavoidable even if we restrict ourselves to deciding FO logic on trees.

In this paper our aim is to investigate if it is possible to go around this harsh lower bound by restricting the considered class of input graphs further. In other words, we are looking for meta-theorems which would imply that all of FO or MSO logic can be solved in time not only linear in the size of the graph, but also depending more reasonably on the secondary parameters, if we are willing to give up some of the generality of the class of bounded-treewidth graphs. We concentrate on two graph classes: graphs of bounded vertex cover and graphs of bounded max-leaf number. We note that the investigation of the existence of stronger meta-theorems for these classes has been posed explicitly as an open problem by Fellows in [7].

Though graphs of bounded vertex cover or max-leaf number are considerably more restricted than bounded treewidth graphs, these classes are still interesting from the algorithmic point of view and the complexity of hard problems parameterized by vertex cover or max-leaf number has been investigated in the past ([9], [8]). Furthermore, as mentioned, strong lower bounds are known to apply to slightly more general classes: for bounded feedback vertex set and bounded pathwidth graphs even FO logic is non-elementary, while even for binary trees (thus for graphs of bounded treewidth and max degree) FO logic is at least triply exponential (again by [13]). Bounded vertex cover and bounded max-leaf number evade all these lower bound arguments so it's natural to ask what is exactly the complexity of FO and MSO logic for these classes of graphs?

The main results of this paper show that meta-theorems stronger than Courcelle's can indeed be shown for these classes of graphs. In addition, we show that our meta-theorems for vertex cover cannot be significantly improved under standard complexity assumptions.

Specifically, for the class of graphs of vertex cover bounded by k we show that

– All graph problems expressible with an FO formula ϕ can be solved in time linear in the graph size and singly exponential in k and $|\phi|$.

– All graph problems expressible with an MSO_1 formula ϕ can be solved in time linear in the graph size and doubly exponential in k and $|\phi|$.
– Unless n-variable 3SAT can be solved in time $2^{o(n)}$ (that is, unless the Exponential Time Hypothesis fails), then no $f(k,\phi)\cdot poly(|G|)$ algorithm exists to decide MSO logic on graphs for any $f(k,\phi) = 2^{2^{o(k+|\phi|)}}$.
– Unless FPT=W[1], there is no algorithm which can decide if an FO formula ϕ with q quantifiers holds in a graph G of vertex cover k in time $f(k,q)n^c$, for any $f(k,q) = 2^{O(k+q)}$.

Furthermore, for the class of graphs of max-leaf number bounded by k we show that

– All graph problems expressible with an FO formula ϕ can be solved in time linear in the graph size and singly exponential in k and $|\phi|$.

Our upper bounds rely on techniques different from the standard dynamic programming on decompositions usually associated with treewidth. For max-leaf number we rely on the characterization of bounded max-leaf number graphs from [16] also used heavily in [8] and the fact that FO logic has limited counting power in paths. For vertex cover we exploit an observation that for FO logic two vertices that have the same neighbors are "equivalent" in a sense we will make precise. We state our results in this case in terms of a new graph "width" parameter that captures this graph property more precisely than bounded vertex cover. We call the new parameter neighborhood diversity, and the upper bounds for vertex cover follow by showing that bounded vertex cover is a special case of bounded neighborhood diversity. Our essentially matching lower bounds on the other hand are shown for vertex cover. In the last section of this paper we prove some additional results for neighborhood diversity, beyond the algorithmic meta-theorems of the rest of the paper, which we believe indicate that neighborhood diversity might be a graph structure parameter of independent interest and that its algorithmic and graph-theoretic properties may merit further investigation.

Due to space constraints, some of the proofs have been omitted and will appear in the full version of this paper.

2 Definitions and Preliminaries

Model Checking, FO and MSO Logic. In this paper we will describe algorithmic meta-theorems, that is, general methods for solving all problems belonging in a class of problems. However, the presentation is simplified if one poses this approach as an attack on a single problem, the model checking problem. In the model checking problem we are given a logic formula ϕ, expressing a graph property, and a graph G, and we must decide if the property described by ϕ holds in G. In that case, we write $G \models \phi$. Clearly, if we can describe an efficient algorithm for model checking for a specific logic, this will imply the existence of efficient algorithms for all problems expressible in this logic. Let us now give more details about the logics we will deal with and the graphs which will be our input instances.

Our universe of discourse will be labeled, colored graphs. Specifically, we assume that the first part of the input is an undirected graph $G(V, E)$, a set of labels L, each associated with a vertex of V and a set of subsets of V, $\mathcal{C} = \{C_1, C_2, \ldots, C_c\}$, which we refer to as color classes. The interesting case here is unlabeled, uncolored graphs (that is, $L = \mathcal{C} = \emptyset$), but the additional generality in the definition of the problem makes it easier to describe a recursive algorithm.

The formulas of FO logic are those which can be constructed using vertex variables, denoted usually by $x_i, y_i, \ldots$, vertex labels denoted by l_i, color classes denoted by C_i, the predicates $E(x_i, x_j)$, $x_i \in C_j$, $x_i = x_j$ operating on vertex variables or labels, standard propositional connectives and the quantifiers $\exists, \forall$ operating on vertex variables. The semantics are defined in the usual way, with the $E()$ predicate being true if $(x_i, x_j) \in E$.

For MSO logic the additional propery is that we now introduce set variables denoted by X_i and allow the quantifiers and the $\in$ predicate to operate on them. The semantics are defined in the obvious way.

If the set variables are allowed to range over sets of vertices only then the logic is sometimes referred to as MSO_1. A variation is MSO_2 logic, where one is also allowed to use set variables that range overs sets of edges. To accomodate for this case one also usually modifies slightly the definition of FO formulas to allow edge variables and an incidence predicate $I(v, e)$ which is true is true if edge e is incident on vertex v.

Bounded Vertex Cover and neighborhood diversity. We will work extensively with graphs of bounded vertex cover, that is, graphs for which there exists a small set of vertices whose removal also removes all edges. We will usually denote the size of a graph's vertex cover by k. Note that there exist linear-time FPT algorithms for finding an optimal vertex cover in graphs where k is small (see e.g. [2]).

Our technique relies on the fact that in a graph of vertex cover k, the vertices outside the vertex cover can be partitioned into at most 2^k sets, such that all the vertices in each set have exactly the same neighbors outside the set and each set contains no edges inside it. Since we do not make use of any other special property of graphs of small vertex cover, we are motivated to define a new graph parameter, called neighborhood diversity, which intuitively seems to give the largest graph family to which we can apply our method in a straightforward way.

Definition 1. *We will say that two vertices v, v' of a graph $G(V, E)$ have the same type iff they have the same colors and $N(v) \setminus \{v'\} = N(v') \setminus \{v\}$.*

Definition 2. *A colored graph $G(V, E)$ has neighborhood diversity at most w, if there exists a partition of V into at most w sets, such that all the vertices in each set have the same type.*

Lemma 1. *If an uncolored graph has vertex cover at most k, then it has neighborhood diversity at most $2^k + k$.*

In Section 7 we will show that neighborhood diversity can be computed in polynomial time and also prove some results which indicate it may be an interesting parameter in its own right. However, until then our main focus will be graphs of bounded vertex cover. We will prove most of our algorithmic results in terms of neighborhood diversity and then invoke Lemma 1 to obtain our main objective. We will usually assume that a partition of the graph into sets with the same neighbors is given to us, because otherwise one can easily be found in linear time by using the mentioned linear-time FPT algorithm for vertex cover and Lemma 1.

Bounded Max-Leaf Number. We say that a connected graph G has max-leaf number at most l if no spanning tree of G has more than l leaves. The algorithmic properties of this class of graphs have been investigated in the past [6,10,8]. In this paper we rely heavily on a characterization of bounded max-leaf graphs by Kleitman and West [16] which is also heavily used in [8].

Theorem 1. *[16] If a graph G has max-leaf number at most l, then G is a subdivision of a graph on $O(l)$ vertices.*

What this theorem tells us intuitively is that in a graph $G(V, E)$ with max-leaf number l there exists a set S of $O(l)$ vertices such that $G[V \setminus S]$ is a collection of $O(l^2)$ paths. Furthermore, only the endpoints of the paths can be connected to vertices of S in G.

It is well-known that a graph of max-leaf number at most l has a path decomposition of width at most $2l$. Furthermore, it must have maximum degree at most l. Bounded max-leaf number graphs are therefore a subclass of the intersection of bounded pathwidth and bounded degree graphs (in fact, they are a proper subclass, as witnessed by the existence of say $2 \times n$ grids). Let us mention again that deciding FO logic on binary trees has at least a triply exponential parameter dependence, so the results we present for graphs of bounded max-leaf number can also be seen as an improvement on the currently known results for FO logic on bounded degree graphs, for this more restricted case.

3 FO Logic for Bounded Vertex Cover

In this Section we show how any FO formula can be decided on graphs of bounded vertex cover, with a singly exponential parameter dependence. Our main argument is that for FO logic, two vertices which have the same neighbors are essentially equivalent. We will state our results in the more general case of bounded neighborhood diversity and then show the corresponding result for bounded vertex cover as a corollary.

Lemma 2. *Let $G(V, E)$ be a graph and $\phi(x)$ a FO formula with one free variable. Let $v, v' \in V$ be two distinct unlabeled vertices of G that have the same type. Then $G \models \phi(v)$ iff $G \models \phi(v')$.*

Proof. (Sketch) Recall that the standard way of deciding an FO formula on a graph is, whenever we encounter an existential quantifier to try all possible choices of a vertex for that variable. This creates an n-ary decision tree with height equal to the number of quantifiers in $\phi(x)$. Every leaf corresponds to a choice of vertices for the q quantified variables, which makes the formula true or false. Internal nodes are evaluated as the disjunction (for existential quantifiers) or conjunction (for universal quantifiers) of their children.

It is possible to create a one-to-one correspondence between the trees for $\phi(v)$ and $\phi(v')$, by essentially exchanging v and v', showing that $\phi(v)$ and $\phi(v')$ are equivalent. □

Theorem 2. *Let ϕ be a FO sentence of quantifier depth q. Let $G(V, E)$ be a labeled colored graph with neighborhood diversity at most w and l labeled vertices. Then, there is an algorithm that decides if $G \models \phi$ in time $O((w + l + q)^q \cdot |\phi|)$.*

Corollary 1. *There exists an algorithm which, given a FO sentence ϕ with q variables and an uncolored, unlabeled graph G with vertex cover at most k, decides if $G \models \phi$ in time $2^{O(kq+q \log q)}$.*

Thus, the running time is (only) singly exponential in the parameters, while a straightforward observation that bounded vertex cover graphs have bounded treewidth and an application of Courcelle's theorem would in general have a non-elementary running time. Of course, a natural question to ask now is whether it is possible to do even better, perhaps making the exponent linear in the parameter, which is $(k + q)$. As we will see later on, this is not possible if we accept some standard complexity assumptions.

4 FO Logic for Bounded Max-Leaf Number

In this section we describe a model checking algorithm for FO logic on graphs of small max-leaf number. Our main tool is the mentioned observation that all but a small fraction of the vertices have degree 2, and therefore (since we assume without loss of generality that the graph is connected) induce paths. We call a maximal set of connected vertices of degree 2 a topo-edge.

Our main argument is that when a topo-edge is very long (exponentially long in the number of quantifiers of the first-order sentence we are model checking) its precise length does not matter.

To make this more precise, let us first define a similarity relation on graphs.

Definition 3. *Let G_1, G_2, be two graphs. For a given q we will say that G_1 and G_2 are q-similar and write $G_1 \sim_q G_2$ iff G_1 contains a topo-edge of order at least 2^{q+1} consisting of unlabeled vertices, call it P, and G_2 can be obtained from G_1 by contracting one of the edges of P. We denote the transitive closure of the relation $\sim_q$ as $\sim_q^*$.*

Our main technical tool is now the following lemma.

Lemma 3. *Let ϕ be a FO formula with q quantifiers. Then, for any two graphs G_1, G_2 if $G_1 \sim_q G_2$ then $G_1 \models \phi$ iff $G_2 \models \phi$. Therefore, if $G_1 \sim_q^* G_2$ then $G_1 \models \phi$ iff $G_2 \models \phi$.*

Now we are ready to state our main result of this section.

Theorem 3. *Let G be a graph on n vertices with max-leaf number k and ϕ a FO formula with q quantifiers. Then, there exists an algorithm for deciding if $G \models \phi$ running in time $poly(n) + 2^{O(q^2 + q \log k)}$.*

Proof. By applying Theorem 1 we know that G can be partitioned into a set of at most $O(k)$ vertices of degree at least 3 and a collection of paths. By applying Lemma 3 we know that there exists a G' such that $G \sim_q^* G'$ and G' consists of the same $O(k)$ vertices of degree at least 3 and at most $O(k^2)$ paths whose length is at most 2^{q+1}. Of course, G' can be found in time polynomial in n.

Now, we can apply the straightforward algorithm to model check ϕ on G'. For every quantifier we have at most $O(k + q) + O(k(k + q)2^{q+1})$ choices, which is $2^{O(q+\log k)}$. Exhausting all possibilities for each vertex gives the promised running time. □

5 MSO Logic for Bounded Vertex Cover

First, let us state a helpful extension of the results of the Section 3. From the following Lemma it follows naturally that the model checking problem for MSO_1 logic on bounded vertex cover graphs is in XP, that is, solvable in polynomial time for constant ϕ and k, but our objective later on will be to do better.

Lemma 4. *Let $\phi(X)$ be an MSO_1 formula with a free set variable X. Let G be a graph and S_1, S_2 two sets of vertices of G such that all vertices of $(S_1 \setminus S_2) \cup (S_2 \setminus S_1)$ are unlabeled and have the same type and furthermore $|S_1 \setminus S_2| = |S_2 \setminus S_1|$. Then $G \models \phi(S_1)$ iff $G \models \phi(S_2)$.*

Our main tool in this section is the following lemma.

Lemma 5. *Let $\phi(X)$ be an MSO_1 formula with one free set variable X, q_V quantified vertex variables and q_S quantified set variables. Let G be a graph and S_1, S_2 two sets of vertices of G such that all vertices of $(S_1 \setminus S_2) \cup (S_2 \setminus S_1)$ are unlabeled and belong in the same type T. Suppose that both $|S_1 \cap T|$ and $|S_2 \cap T|$ fall in the interval $[2^{q_S} q_V, |T| - 2^{q_S} q_V - 1]$. Then $G \models \phi(S_1)$ iff $G \models \phi(S_2)$.*

Proof. (Sketch) We can assume without loss of generality that $S_1 \subseteq S_2$, thanks to Lemma 4. In fact, we may assume that $S_2 = S_1 \cup \{u\}$ for some vertex u and repeated applications of the same argument yield the claimed result.

Our argument is that the truth of $\phi(S_1)$ and $\phi(S_2)$ can be decided by an algorithm which checks for each set variable every combination of sizes that its intersection has with each type and for each vertex variable one representative of each type. If u is never picked as a representative then the algorithm must give the same answer for $\phi(S_1)$ and $\phi(S_2)$. This can be guaranteed if the type u belongs in always has more than q_V vertices. Every time the algorithm picks a value for a set variable, the type u belongs in is partitioned in two. Since the only thing that matters to the algorithm is the size of the set's intersection with u's type, we are free to select a set that puts u in the larger of the two new sub-types. Because $S_1 \cap T$ and $S_2 \cap T$ have constrained sizes, we can always guarantee that u is never selected. $\square$

Theorem 4. *There exists an algorithm which, given a graph G with l labels, neighborhood diversity at most w and an MSO$_1$ formula ϕ with at most q_S set variables and q_V vertex variables, decides if $G \models \phi$ in time $2^{O\left(2^{q_S}(w+l)q_S^2 q_V \log q_V\right)} \cdot |\phi|$.*

Corollary 2. *There exists an algorithm which, given an MSO$_1$ sentence ϕ with q variables and an uncolored, unlabeled graph G with vertex cover at most k, decides if $G \models \phi$ in time $2^{2^{O(k+q)}}$.*

Again, this gives a dramatic improvement compared to Courcelle's theorem, though exponentially worse than the case of FO logic. This is an interesting point to consider because for treewidth there does not seem to be any major difference between the complexities of model checking FO and MSO$_1$ logic.

The natural question to ask here is once again, can we do significantly better? For example, perhaps the most natural question to ask is, is it possible to solve this problem in $2^{2^{o(k+q)}}$? As we will see later on, the answer is no, if we accept some standard complexity assumptions.

Finally, let us briefly discuss the case of MSO$_2$ logic. In general this logic is more powerful than MSO$_1$, so it is not straightforward to extend Theorem 4 in this case. However, if we are not interested in neighborhood diversity but just in vertex cover we can observe that all edges in a graph with vertex cover of size k have one of their endpoints in one of the k vertices of the vertex cover. Thus, any edge set X can be written as the union of k edge sets. In turn, each of these k edge sets can easily be replaced by vertex sets, without loss of information, since we already know one of the endpoints of each of these edges. Using this trick we can replace every edge set variable in an MSO$_2$ sentence with k vertex set variables, thus obtaining a $2^{2^{O(kq)}}$ algorithm for MSO$_2$ logic on graphs of bounded vertex cover.

6 Lower Bounds

In this Section we will prove some lower bound results for the model checking problems we are dealing with. Our proofs rely on a construction which reduces SAT to a model checking problem on a graph with small vertex cover.

Given a propositional 3-CNF formula ϕ_p with n variables and m clauses, we want to construct a graph G that encodes its structure, while having a small vertex cover. The main problem is encoding numbers up to n with graphs of small vertex cover but this can easily be achieved by using the binary representation of numbers.

We begin constucting a graph by adding $7 \log n$ vertices, call them $u_{(i,j)}, 1 \leq i \leq 7, 1 \leq j \leq \log n$. Add all edges of the form $(u_{(i,j)}, u_{(k,j)})$ (so we now have $\log n$ disjoint copies of K_7). Let $N_i = \{u_{(i,j)} \mid 1 \leq j \leq \log n\}$.

For every variable x_i in ϕ_p add a new vertex to the graph, call it v_i. Define for every number i the set $X(i) = \{j \mid \text{the j-th bit of the binary representation of i is 1}\}$. Add the edges $(v_i, u_{(1,j)}), j \in X(i)$, that is connect every variable vertex with the vertices of N_1 that correspond to the binary representation of its index. Let $U = \{v_i \mid 1 \leq i \leq n\}$ be the vertices corresponding to variables.

For every clause c_i in ϕ_p add a new vertex to the graph, call it w_i. If the first literal in c_i is a positive variable x_k then add the edges $(w_i, u_{(2,j)}, j \in X(k)$. If the first literal is a negated variable $\neg x_k$, add the edges $(w_i, u_{(3,j)}, j \in X(k)$. Proceed in a similar way for the second and third literal, that is, if the second literal is positive connect w_i with the vertices that correspond to the binary representation of the variable in N_4, otherwise in N_5. For the third literal do the same with N_6 or N_7. Let $W = \{w_i \mid 1 \leq i \leq m\}$ be the vertices corresponding to clauses.

Finally, set the color classes to be $\{N_1, N_2, \ldots, N_7, U, W\}$.

Now, looking at the graph it is easy to see if a vertex v_i corresponds to a variable that appears positive in the clause represented by a vertex w_i. They must satisfy the formula

$$pos(v_i, w_j) = \bigvee_{k=2,4,6} \forall x(x \in N_1 \rightarrow \exists y((E(v_i, x) \leftrightarrow E(w_j, y)) \land y \in N_k \land E(x, y)))$$

It is not hard to define $neg(v_i, w_j)$ in a similar way. Now it is straight-forward to check if ϕ_p was satisfiable:

$$\phi = \exists S(\forall x x \in S \rightarrow x \in U) \land (\forall w w \in W \rightarrow \exists x x \in U \land \\ ((pos(x, w) \land x \in S) \lor (neg(x, w) \land x \notin S)))$$

Clearly, ϕ holds in the constructed graph iff ϕ_p is satisfiable. S corresponds to the set of variables set to true in a satisfying assignment. Let us also briefly remark that it is relatively easy to eliminate the colors and labels from the construction above, therefore the lower bounds given below apply to the natural form of the problem.

Lemma 6. *$G \models \phi$ iff ϕ_p is satisfiable. Furthermore, ϕ has size $O(1)$ and G has a vertex cover of size $O(\log n)$.*

Theorem 5. *Let ϕ be a MSO formula with q quantifiers and G a graph with vertex cover k. Then, unless 3-SAT can be solved in time $2^{o(n)}$, there is no algorithm which decides if $G \models \phi$ in time $O(2^{2^{o(k+q)}} \cdot poly(n))$.*

Theorem 6. *Let ϕ be a FO formula with q_v vertex quantifiers and G a graph with vertex cover k. Then, unless FPT=W[1], there is no algorithm which decides if $G \models \phi$ in time $O(2^{O(k+q_v)} \cdot poly(n))$.*

7 Neighborhood Diversity

In this Section we give some general results on the new graph parameter we have defined, neighborhood diversity. We will use $nd(G), tw(G), cw(G)$ and $vc(G)$ to denote the neighborhood diversity, treewidth, cliquewidth and minimum vertex cover of a graph G. We will call a partition of the vertex set of a graph G into w sets such that all vertices in every set share the same type a neighborhood partition of width w.

First, some general results

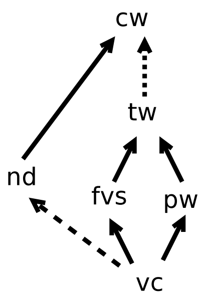
Theorem 7. *1. Let $V_1, V_2, \ldots, V_w$ be a neighborhood partition of the vertices of a graph $G(V,E)$. Then each V_i induces either a clique or an independent set. Furthermore, for all i, j the graph either includes all possible edges from V_i to V_j or none.*

2. For every graph G we have $nd(G) \leq 2^{vc(G)} + vc(G)$ and $cw(G) \leq nd(G) + 1$. Furthermore, there exist graphs of constant treewidth and unbounded neighborhood diversity and vice-versa.

3. There exists an algorithm which runs in polynomial time and given a graph $G(V,E)$ finds a neighborhood partition of the graph with minimum width.

Taking into account the observations of Theorem 7 we summarize what we know about the graph-theoretic and algorithmic properties of neighborhood diversity and related measures in Figure 1.

There are several interesting points to make here. First, though our work is motivated by a specific goal, beating the lower bounds that apply to graphs of bounded treewidth by concentrating on a special case, it seems that what we have achieved is at least somewhat better; we have managed to improve the algorithmic meta-theorems that were known by focusing on a class which is not



	FO	MSO	MSO$_2$
Cliquewidth	$tow(w)$	$tow(w)$	$tow(w)$
Treewidth	$tow(w)$	$tow(w)$	$tow(w)$
Vertex Cover	$2^{O(w)}$	$2^{2^{O(w)}}$	$2^{2^{O(w)}}$
Neighborhood Diversity	$poly(w)$	$2^{O(w)}$	Open

Fig. 1. A summary of the relations between neighborhood diversity and other graph widths. Included are cliquewidth, treewidth, pathwidth, feedback vertex set and vertex cover. Arrows indicate generalization, for example bounded vertex cover is a special case of bounded feedback vertex set. Dashed arrows indicate that the generalization may increase the parameter exponentially, for example treewidth w implies cliquewidth at most 2^w. The table summarizes the best known model checking algorithm's dependence on each width for the corresponding logic.

necessarily smaller than bounded treewidth, only different. However, our class is a special case of another known width which generalizes treewidth as well, namely cliquewidth. Since the lower bound results which apply to treewidth apply to cliquewidth as well, this work can perhaps be viewed more appropriately as an improvement on the results of [4] for bounded cliquewidth graphs.

Second, is the case of MSO_2 logic. The very interesting hardness results shown in [11] demonstrate that the tractability of MSO_2 logic is in a sense the price one has to pay for the additional generality that cliquewidth provides over treewidth. It is natural to ask if the results of [11] can be strengthened to apply to neighborhood diversity or MSO_2 logic can be shown tractable parameterized by neighborhood diversity.

Though we cannot yet fully answer the above question related to MSO_2, we can offer some first indications that this direction might merit further investigation. In [11] it is shown that MSO_2 model checking is not fixed-parameter tractable when the input graph's cliquewidth is the parameter by considering three specific MSO_2-expressible problems and showing that they are W-hard. The problems considered are Hamiltonian cycle, Graph Chromatic Number and Edge Dominating Set. We can show that these three problems can be solved efficiently on graphs of small neighborhood diversity. Since small neighborhood diversity is a special case of small cliquewidth, where these problems are hard, this result could be of independent interest.

Theorem 8. *Given a graph G whose neighborhood diversity is w, there exist algorithms running in time $O(f(w) \cdot poly(|G|))$ that decide Hamiltonian cycle, Graph Chromatic Number and Edge Dominating Set.*

8 Conclusions and Open Problems

In this paper we presented algorithmic meta-theorems which improve the running times implied by previously known meta-theorems for more restricted inputs. In this way we have partially explored the trade-off which can be achieved between running time and generality. This is an interesting area for further investigations and much more can be done.

For bounded max-leaf number the complexity of MSO logic is unknown. Quite likely, it is possible to improve upon the Courcelle's theorem for this case as well, but the problem remains open. Also, it would be nice to obtain a lower bound for FO logic in this case showing that it is impossible to achieve $2^{o(q^2)}$, i.e. that the exponent must be quadratic. For neighborhood diversity the most interesting open problem is the complexity of MSO_2.

Going further, it would also make sense to investigate whether restricting the model checking problem to graphs of bounded vertex cover or max-leaf number can also allow us to solve logics wider than MSO_2. Some indications that this may be possible are given in [9].

References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. J. Algorithms 12(2), 308–340 (1991)
2. Chen, J., Kanj, I.A., Xia, G.: Improved parameterized upper bounds for vertex cover. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 238–249. Springer, Heidelberg (2006)
3. Courcelle, B.: The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. Inf. Comput. 85(1), 12–75 (1990)
4. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. Theory Comput. Syst. 33(2), 125–150 (2000)
5. Dawar, A., Grohe, M., Kreutzer, S., Schweikardt, N.: Approximation schemes for first-order definable optimisation problems. In: LICS, pp. 411–420. IEEE Computer Society, Los Alamitos (2006)
6. Estivill-Castro, V., Fellows, M.R., Langston, M.A., Rosamond, F.A.: FPT is P-Time Extremal Structure I. In: Broersma, H., Johnson, M., Szeider, S. (eds.) ACiD. Texts in Algorithmics, vol. 4, pp. 1–41. King's College, London (2005)
7. Fellows, M.R.: Open problems in parameterized complexity. In: AGAPE spring school on fixed parameter and exact algorithms (2009)
8. Fellows, M.R., Lokshtanov, D., Misra, N., Mnich, M., Rosamond, F.A., Saurabh, S.: The Complexity Ecology of Parameters: An Illustration Using Bounded Max Leaf Number. Theory Comput. Syst. 45(4), 822–848 (2009)
9. Fellows, M.R., Lokshtanov, D., Misra, N., Rosamond, F.A., Saurabh, S.: Graph layout problems parameterized by vertex cover. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 294–305. Springer, Heidelberg (2008)
10. Fellows, M.R., Rosamond, F.A.: The Complexity Ecology of Parameters: An Illustration Using Bounded Max Leaf Number. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 268–277. Springer, Heidelberg (2007)
11. Fomin, F.V., Golovach, P.A., Lokshtanov, D., Saurabh, S.: Clique-width: on the price of generality. In: Mathieu, C. (ed.) SODA, pp. 825–834. SIAM, Philadelphia (2009)
12. Frick, M., Grohe, M.: Deciding first-order properties of locally tree-decomposable structures. J. ACM 48(6), 1184–1206 (2001)
13. Frick, M., Grohe, M.: The complexity of first-order and monadic second-order logic revisited. Ann. Pure Appl. Logic 130(1-3), 3–31 (2004)
14. Grohe, M.: Logic, graphs, and algorithms. Electronic Colloquium on Computational Complexity (ECCC) 14(091) (2007)
15. Hlinený, P., il Oum, S., Seese, D., Gottlob, G.: Width parameters beyond tree-width and their applications. Comput. J. 51(3), 326–362 (2008)
16. Kleitman, D., West, D.: Spanning trees with many leaves. SIAM Journal on Discrete Mathematics 4, 99 (1991)
17. Kreutzer, S., Tazari, S.: On brambles, grid-like minors, and parameterized intractability of monadic second order logic. In: SODA (2010)

Determining Edge Expansion and Other Connectivity Measures of Graphs of Bounded Genus

Viresh Patel[*]

School of Engineering and Computing Sciences, Durham University,
Science Laboratories, South Road, Durham DH1 3LE, U.K.
viresh.patel@dur.ac.uk

Abstract. In this paper, we show that for an n-vertex graph G of genus g, the edge expansion of G can be determined in time $n^{O(g^2)}$. We show that the same is true for various other similar measures of edge connectivity.

1 Introduction

1.1 Background and Motivation

Edge expansion (known also as the minimum cut quotient, the isoperimetric number, or the flux of a graph) is a well-studied notion in graph theory and arises in several contexts of discrete mathematics and theoretical computer science. These include the explicit construction of expander graphs, the analysis of certain randomised algorithms, and graph partitioning problems. In this paper, we are concerned with giving an exact algorithm for determining the edge expansion (and other similar measures) of graphs embedded on surfaces.

Throughout, we use the term graph to mean multigraph without loops, unless otherwise stated. For a graph $G = (V, E)$ and $e \in E$, we write $e = ab$ to mean that the vertices a and b are the end points of e.

For S a nonempty proper subset of V and $\bar{S}$ its complement, we define

$$[S, \bar{S}]_G = \{e \in E : \ e = ab, \ a \in S, \ b \in \bar{S}\},$$

which we call an *edge-cut* of G. (The subscript is dropped when it clear which graph we are referring to.) For a cut $[S, \bar{S}]$ of a graph $G = (V, E)$, define the *balance* of the cut to be $b(S, \bar{S}) := \min(|S|, |\bar{S}|)/|V|$. Note that the balance of a cut is a real number in the interval $(0, \frac{1}{2}]$. Two well-known graph cut problems, which take into account the balance of cuts, are the *minimum quotient cut* problem and the *sparsest cut* problem. These ask respectively to minimize the *cut quotient* $q(S, \bar{S})$ and the *cut density* $d(S, \bar{S})$ over all cuts $[S, \bar{S}]$ of a graph G, where

$$q(S, \bar{S}) = \frac{|[S, \bar{S}]|}{b(S, \bar{S})} \quad \text{and} \quad d(S, \bar{S}) = \frac{|[S, \bar{S}]|}{b(S, \bar{S})(1 - b(S, \bar{S}))}.$$

[*] Supported by EPSRC grant EP/F064551/1.

Both q and d penalise unbalanced cuts, although q does so to a greater extent than d. Note also that q and d differ by at most a factor of 2.

Problems such as the minimum quotient cut problem and the sparsest cut problem underlie many divide and conquer algorithms [16], and find applications in VLSI layout problems, packet routing in distributed networking, clustering, and so on. Unfortunately, for general graphs, finding a minimum quotient cut or a sparsest cut is known to be NP-hard [6,10]. Thus there are two possible ways of developing efficient algorithms for these problems: either by considering approximation algorithms or by restricting attention to certain graph classes. There has been much research done in finding approximation algorithms for these problems. Here, we mention only the seminal paper of Leighton and Rao [9] giving a polynomial-time $O(\log n)$-approximation algorithm for the minimum quotient cut problem, and the significant improvement in the approximation factor to $O(\sqrt{\log n})$ in a paper of Arora, Rao, and Vazirani [2]. On the hardness side, Ambühl et al. [1] proved that the sparsest cut problem admits no polynomial-time approximation scheme unless NP-hard problems can be solved in randomized subexponential time.

We approach the problems of minimum quotient cut and sparsest cut from the perspective of developing exact polynomial-time algorithms for restricted graph classes. Such approaches have not received as much attention in recent years as the development of approximation algorithms, but we hope this paper will take a step towards sparking interest.

Bonsma [3] gave polynomial-time algorithms for finding sparsest cuts of unit circular graphs and cactus graphs. Park and Phillips [14], building on the work of Rao [15], gave a polynomial-time algorithm for determining the minimum quotient cut (as it has been defined above) of planar graphs. Given that many planar-graph algorithms have been adapted for generalizations of planar graphs – see for example the introduction to [4] and the references therein – surprisingly little is known about the complexity of computing minimum quotient cuts or sparsest cuts for generalizations of planar graphs. Here, we generalize the algorithm of Park and Phillips to give the first exact polynomial-time algorithm for determining minimum quotient cuts and sparsest cuts of bounded-genus graphs.

1.2 Results

Before we state our result precisely, we give a generalization of the minimum quotient cut and sparsest cut. Notice that the denominators for both the cut quotient q and the cut density d are concave and increasing functions of $b(S, \bar{S})$ on the interval $[0, \frac{1}{2}]$. For any concave, increasing function $f : [0, \frac{1}{2}] \to [0, \infty)$ and a cut $[S, \bar{S}]$ of a graph G, we define

$$d_G^f(S, \bar{S}) = \frac{|[S, \bar{S}]|}{f\big(b(S, \bar{S})\big)},$$

and we define

$$d^f(G) = \min d_G^f(S, \bar{S}),$$

where the minimum is taken over all cuts $[S, \bar{S}]$ of G. Any cut $[S, \bar{S}]$ that minimizes d_G^f is referred to as an *f-sparsest cut* of G.

Let $f : [0, \frac{1}{2}] \to [0, \infty)$ be a fixed concave, increasing function that is computable in polynomial time on the rationals, and let g be a fixed non-negative integer. The input for our algorithm is an n-vertex undirected multigraph G of genus g. Our algorithm computes an f-sparsest cut of G in time $O(n^{2g^2+4g+7})$.

1.3 Overview and Techniques

In this section we give an informal overview of our methods. Our methods extend those of Park and Phillips [14] and combine them with surface homology techniques, used for example in [4].

A generalisation of an averaging argument given by Mohar [11] shows that, given a graph G, there exists a sparsest cut $[S, \bar{S}]$ of G that is *minimal*, i.e. a cut where the graphs induced by G on S and $\bar{S}$ are both connected. This extends easily to f-sparsest cuts, where f is a concave increasing function.

There is a standard correspondence between the cuts of a planar graph G and the cycles of its dual $D(G)$: the minimal cuts of G correspond precisely to the cycles of $D(G)$, and the size of a cut in G is equal to the length of its corresponding cycle in $D(G)$. One can similarly construct a dual graph $D(G)$ for a graph G embedded on a surface; however the correspondence between minimal cuts of G and cycles of $D(G)$ is not quite so simple. Roughly, for a graph G of genus g, a minimal cut of G corresponds to a union of at most $g + 1$ cycles of $D(G)$, but the reverse does not hold: a union of at most $g+1$ cycles in $D(G)$ does not necessarily correspond to a cut in G. Using the surface embedding of G, we construct a function Θ from the set of oriented edges of $D(G)$ to $\mathbb{Z}^{2g}$ with the following property: summing Θ around the oriented edges of a union of cycles of $D(G)$ gives the zero vector if and only if that union of cycles corresponds to a (certain generalization of a) cut of G.

Extending and simplifying an idea from [14], we also construct a weighting $\hat{w}$ on the set of oriented edges of $D(G)$ with the following property: if a union of cycles in $D(G)$ corresponds to a cut in G, then summing $\hat{w}$ around the oriented edges of cycles in the union essentially gives the balance of $[S, \bar{S}]$.

Using $D(G)$, Θ, and $\hat{w}$, we construct a type of covering graph H, again extending an idea in [14]. For each fixed value $\mathbf{v}$ and k of Θ and $\hat{w}$, we can use H to find a shortest cycle in $D(G)$ whose Θ-value is $\mathbf{v}$ and whose $\hat{w}$-value is k. Such a shortest cycle of $D(G)$ corresponds to a shortest path in H between suitable vertices.

By repeatedly applying a shortest-path algorithm to H, we obtain, for every $\mathbf{v}$ and k (in a suitable range), a shortest cycle of $D(G)$ whose Θ-value is $\mathbf{v}$ and whose $\hat{w}$-value is k. We construct the set X of every union of at most $g + 1$ of these shortest cycles. The size of X is $n^{O(g^2)}$, and we show that at least one element of X corresponds to an f-sparsest cut of G.

2 Preliminaries

Due to limited space, we only state the various lemmas we require for our algorithm; however, complete proofs will be given in the full version of this paper.

Throughout, rather than working with functions $f : [0, \frac{1}{2}] \to [0, \infty)$ that are concave and increasing, we work instead with functions $f : [0, 1] \to [0, \infty)$ that are concave and increasing on $[0, \frac{1}{2}]$ and have the property that $f(x) = f(1 - x)$ for all $x \in [0, 1]$. We work with these functions purely for the convenience of having, for any cut $[S, \bar{S}]$ of G, that

$$f(b(S, \bar{S})) = f(|S|/|V|) = f(|\bar{S}|/|V|).$$

For the algorithm, we assume that f can be computed in polynomial time on the rationals.

The description and the proof of correctness of our algorithm is most conveniently and naturally expressed in the language of surface homology. Through the remainder of this section, we introduce the necessary concepts, keeping our treatment as simple and self-contained as possible. One can find more comprehensive treatments in e.g. [7,8]

Although we are only concerned with undirected graphs when determining f-sparsest cuts, we will need to orient edges of our graphs through the course of our proofs and algorithm. Each edge $e = ab$ of a graph $G = (V, E)$ has two orientations, namely (a, e, b) and (b, e, a). The two orientations are denoted $\overrightarrow{e}$ and $\overleftarrow{e}$, although we cannot say which is which in general. Given a set E of edges, we write $\overrightarrow{E}$ for the set of their orientations, two for each edge. The *edge space* of G, denoted $\mathcal{E}(G)$, is the free abelian group over $\overrightarrow{E}$ modulo the relation that $\overleftarrow{e} = -\overrightarrow{e}$. Thus, for each $\rho \in \mathcal{E}(G)$, we can express ρ uniquely as

$$\rho = \sum_{\overrightarrow{e} \in \overrightarrow{E}} \lambda_{\overrightarrow{e}} \overrightarrow{e},$$

where $\lambda_{\overrightarrow{e}} \in \mathbb{Z}$ for all $\overrightarrow{e} \in \overrightarrow{E}$ and $\min(\lambda_{\overrightarrow{e}}, \lambda_{\overleftarrow{e}}) = 0$. We define

$$|\rho| = \sum_{\overrightarrow{e} \in \overrightarrow{E}} \lambda_{\overrightarrow{e}}.$$

Thus $|\rho|$ in a sense counts the number of edges in ρ.

The *cut space* $\mathcal{T}(G)$ of $G = (V, E)$, which is a subgroup of $\mathcal{E}(G)$, is defined as follows. For a cut $[S, \bar{S}]$ of G, define

$$\overrightarrow{[S, \bar{S}]} = \sum_{\substack{ab = e \in E \\ a \in S,\ b \notin S}} (a, e, b).$$

Then $\mathcal{T}(G)$ is the subgroup of $\mathcal{E}(G)$ generated by $\{\overrightarrow{[S, \bar{S}]} : S \subseteq V\}$.

The next lemma shows that, by suitably assigning weights to oriented edges of a graph, we can determine the balance of a cut simply by summing the weights of the edges in the (oriented) cut.

Lemma 1. *Let $G = (V, E)$ be a connected graph and let $v \in V$ be some fixed vertex. There exists a function $w : \overrightarrow{E} \to \mathbb{Z}$ with the following properties.*

(i) For every $\overrightarrow{e} \in \overrightarrow{E}$, we have $w(\overleftarrow{e}) = -w(\overrightarrow{e})$.
(ii) For every $\overrightarrow{e} \in \overrightarrow{E}$, we have $|w(\overrightarrow{e})| \leq |V|$.
(iii) For every S satisfying $v \in S \subseteq V$, we have that

$$w(\overrightarrow{[S, \bar{S}]}) := \sum_{(a,e,b) \in \overrightarrow{[S, \bar{S}]}} w(a, e, b) = |\bar{S}|.$$

Furthermore, a function satisfying the above properties can be constructed in linear time.

Remark 1. The function w described in Lemma 1 can be extended to a homomorphism $w : \mathcal{E}(G) \to \mathbb{Z}$ because of property (i).

Throughout, w will be a homomorphism from $\mathcal{E}(G)$ to $\mathbb{Z}$ satisfying the properties of Lemma 1.

Recall that the domain for the function d_G^f is the set of all cuts of G. It turns out that it is necessary to extend d_G^f in a natural way to a function on $\mathcal{T}(G)$. Minimizing d_G^f over $\mathcal{T}(G)$ will be equivalent to minimizing it over the set of cuts of G. Although $\mathcal{T}(G)$ is an infinite set, we will eventually look to minimize d_G^f over a suitable finite subset of $\mathcal{T}(G)$.

For each $\phi \in \mathcal{T}(G)$, we define

$$d_G^f(\phi) := \frac{|\phi|}{f(|w(\phi)|/n)}; \tag{1}$$

if $f(|w(\phi)|) = 0$ or $|w(\phi)| > n$, we define $d_G^f(\phi) = \infty$. Note that if $\phi = \overrightarrow{[S, \bar{S}]}$, then $|\phi| = |[S, \bar{S}]|$ and $|w(\phi)|$ is either $|S|$ or $|\bar{S}|$. Hence the function d_G^f defined above extends the definition of d_G^f given in the introduction.

Our next lemma shows that minimizing d_G^f over $\mathcal{T}(G)$ is equivalent to minimizing d_G^f over simple cuts $\overrightarrow{[S, \bar{S}]}$ of G. The proof uses the concavity of f together with a straightforward averaging argument.

Lemma 2. *Let $G = (V, E)$ be a graph. For every $\phi \in \mathcal{T}(G)$, there exists $S \subseteq V$ such that*

$$d_G^f(S, \bar{S}) \leq d_G^f(\phi),$$

so in particular

$$\min_{\phi \in \mathcal{T}(G)} d_G^f(\phi) = d^f(G).$$

A *walk* w in a graph $G = (V, E)$ is specified by giving an alternating sequence of vertices and edges $w = (x_1, e_1, x_2, e_2, \ldots, x_{k-1}, e_{k-1}, x_k)$, where $(x_i, e_i, x_{i+1}) \in \overrightarrow{E}$ for all i. We write $|w|$ for the number of oriented edges (with multiplicity) traversed in w (which in this case is $k - 1$). If $x_1 = x_k$ then w is called a *closed* *walk*. If all (non-oriented) edges of a closed walk w are distinct, then w is called a *circuit* of G. If all the vertices of a closed walk w are distinct (except $x_1 = x_k$), then w is called a *cycle*. A walk in which all vertices are distinct is called a *path*. We write $\overrightarrow{w}$ for the element of $\mathcal{E}(G)$ given by

$$\overrightarrow{w} = \sum_{i=1}^{k-1} (x_i, e_i, x_{i+1});$$

we refer to $\overrightarrow{w}$ as an *oriented* walk, (circuit, etc). Note that for a walk w, we have $|w| \geq |\overrightarrow{w}|$ with equality when w is a circuit.

The *cycle space* $\mathcal{C}(G)$ is the subgroup of $\mathcal{E}(G)$ (redundantly) generated by the oriented cycles $\overrightarrow{c}$ of G. Note that $\mathcal{C}(G)$ contains all oriented closed walks of G.

We now turn our attention to graphs embedded on closed orientable surfaces. Formally, a surface is a compact, connected topological space in which every point of the surface has an open neighbourhood homeomorphic to $\mathbb{R}^2$ or the closed halfplane $\{(x, y) \in \mathbb{R}^2 : y \geq 0\}$. The set of points having halfplane open neighbourhoods is called the boundary of the surface. Every component of the boundary is homeomorphic to the circle S^1. A closed surface is one without boundary. A surface is called orientable if it does not contain a subset (with the subset topology) homeomorphic to the möbius band.

The genus of a connected, orientable surface is the maximum number of cuttings along non-intersecting, closed, simple curves that can be made without disconnecting the surface. It is well known from the classification of surfaces that every closed orientable surface of genus g is homeomorphic to a sphere with g handles. Informally, a graph G can be embedded on a surface Σ if G can be drawn on Σ in such a way that no edge crosses a vertex or another edge, except possibly at its end points. For example, all planar graphs can be embedded on the sphere.

From an algorithmic point of view, one can use *rotation systems* to input or output embeddings of graphs on surfaces. We do not formally define embeddings or rotation systems here because we shall only use graph embeddings indirectly when applying existing algorithms to our problem; instead we refer the reader to [13].

Throughout, we shall only consider cellular embeddings. An embedding of G on Σ is called *cellular* if removing the image of G from Σ leaves a set of topological disks called the *faces* of G. The *genus* of a graph G is defined to be the smallest integer g such that G can be embedded on a closed orientable surface of genus g. If G is a graph of genus g, then every embedding of G on a closed orientable surface of genus g is cellular (Proposition 3.4.1 [13]), and for fixed g, such an embedding can be found in linear time [12]. Euler's Theorem gives the following relationship between the number of vertices n, the number of edges m, the number of faces ℓ, and the number of boundary components b in a cellular embedding of a graph G on a surface Σ of genus g:

$$n - m + \ell + b = 2 - 2g.$$

We now define the *boundary space* of a graph G embedded on Σ. Each oriented edge $\overrightarrow{e}$ of G separates two (possibly equal) faces of G denoted $left(\overrightarrow{e})$ and $right(\overrightarrow{e})$. (The notion of left and right with respect to an oriented edge is well defined for orientable surfaces.) For a face F, the oriented edges $\overrightarrow{e}$ for which $left(\overrightarrow{e}) = F$ taken in order (with appropriate intervening vertices) form a closed walk f around F, which we call the facial walk of F. Let $F_1, \ldots, F_\ell$ be the faces of the embedding and let f_i be the facial walk of F_i. The oriented facial walk $\overrightarrow{f_i}$ of F_i is given by

$$\overrightarrow{f_i} = \sum_{\overrightarrow{e}:\, left(\overrightarrow{e})=F_i} \overrightarrow{e}.$$

Notice that f_i may contain two oppositely oriented edges, but such edges cancel in $\overrightarrow{f_i}$, leaving the sum of oriented edges that form the boundary of F_i. The boundary space $\mathcal{B}(G)$ is defined to be the group generated by $\overrightarrow{f_1}, \ldots, \overrightarrow{f_\ell}$ and is easily seen to be a subgroup of $\mathcal{C}(G)$. Note that the boundary space of G, in contrast to the cycle space and cut space, depends on the embedding of G.

The geometric dual of a graph G cellularly embedded on Σ is denoted by $D(G) = (V', E')$ and is the graph (with cellular embedding on Σ) constructed from G as follows. For each face F of G, a vertex $D(F)$ is placed inside F: these are the vertices of $D(G)$. Each edge e of G has a corresponding edge $D(e)$ in $D(G)$: $D(e) = D(F_1)D(F_2)$, where e is the edge separating the (possibly indistinct) faces F_1 and F_2 (and $D(e)$ crosses e and no other edge of G in the embedding of $D(G)$). Note that G (with its embedding) is a dual of $D(G)$ (with its embedding). Therefore, each vertex v of G corresponds to a face $D(v)$ of $D(G)$. Thus D maps vertices, edges, and faces of G bijectively to faces, edges, and vertices of $D(G)$ respectively. We extend D to map oriented edges of G to oriented edges of $D(G)$ as follows. Given an oriented edge $\overrightarrow{e}$, we set $D(\overrightarrow{e}) = (D(left(\overrightarrow{e})), D(e), D(right(\overrightarrow{e})))$. This, however, reverses the sense of left and right so that $D(D(\overrightarrow{e})) = -\overrightarrow{e}$.

Remark 2. Although G is a loopless graph, $D(G)$ may not be. Nonetheless, all notions introduced so far carry through naturally for loops of embedded graphs.

Since D bijectively maps edges of G to edges of $D(G)$, we see that D can be extended to an isomorphism $D : \mathcal{E}(G) \to \mathcal{E}(D(G))$. We have the following well-known correspondence.

Proposition 1. *The restriction of D to $\mathcal{T}(G)$ gives an isomorphism $\mathcal{T}(G) \to \mathcal{B}(D(G))$.*

Rather than working with $\mathcal{T}(G)$, we can work instead with $\mathcal{B}(D(G))$ using the isomorphism D. Since all boundaries are sums of oriented cycles, we can use shortest-path algorithms to find shortest boundaries in $D(G)$, which if done suitably, can give us f-sparsest cuts in G.

For $\sigma \in \mathcal{E}(D(G))$, define $\hat{w}(\sigma) := \hat{w}(D^{-1}(\sigma))$. Notice that for all $\sigma \in \mathcal{B}(D(G))$, we have $|D^{-1}(\sigma)| = |\sigma|$. Thus, defining

$$\hat{d}^f_{D(G)}(\sigma) = \frac{|\sigma|}{f(|\hat{w}(\sigma)|)}, \tag{2}$$

we have that
$$\min_{\sigma\in\mathcal{B}(D(G))} \hat{d}^f_{D(G)}(\sigma) = \min_{\phi\in\mathcal{T}(G)} d^f_G(\phi) = d^f(G).$$

We now set about trying to minimize $\hat{d}^f_{D(G)}$. Our next lemma says that when minimizing $\hat{d}^f_{D(G)}$, we can restrict attention to elements of $\mathcal{B}(D(G))$ that are the sum of at most $g+1$ oriented circuits (where g is the genus of G). The lemma essentially follows from Euler's Theorem and the fact that there exists an f-sparsest cut that is minimal.

Lemma 3. *Suppose $G=(V,E)$ is a graph cellularly embedded on a surface Σ of genus g. There exists $\sigma\in\mathcal{B}(D(G))$ that minimizes $\hat{d}^f_{D(G)}$ such that $\sigma=\overrightarrow{w_1}+\cdots+\overrightarrow{w_r}$, where $\overrightarrow{w_1},\dots,\overrightarrow{w_r}$ are disjoint oriented circuits of $D(G)$ and $r\le g+1$. Furthermore $m\ge|\sigma|=|\overrightarrow{w_1}|+\cdots+|\overrightarrow{w_r}|$, where $m=|E|$.*

We can easily use shortest-path algorithms to find shortest cycles in a graph. However in this situation, we are required to find a shortest boundary (loosely speaking). We require a simple way of testing whether a cycle is a boundary. This is accomplished using the ideas of homology.

Given a closed, orientable surface Σ of genus g and a point x_0 on Σ, a simple loop at x_0 is a curve $\gamma:[0,1]\to\Sigma$ that is injective except that $\gamma(0)=\gamma(1)=x_0$. A *system of loops* at x_0 is a set of simple loops at x_0 that are pairwise disjoint (except at x_0) with the property that cutting Σ along these loops yields a topological disk. By Euler's formula, every system of loops must consist of $2g$ loops. Let G be an n-vertex graph cellularly embedded on Σ and let x_0 be a vertex. A *combinatorial system of loops* at x_0 is a set of $2g$ closed walks of G through x_0 such that an infinitesimal perturbation of these walks gives a system of loops. Erickson and Whittlesey [5] give a greedy algorithm, which, given a cellular embedding of G on Σ, finds a combinatorial system of loops $c(1),\dots,c(2g)$ in $O(n\log n+gn)$ time.

We define a homomorphism $\Theta_i:\mathcal{C}(D(G))\to\mathbb{Z}$, where, for every $\sigma\in\mathcal{C}(D(G))$, the integer $\Theta_i(\sigma)$ measures the net number of times σ crosses $c(i)$ (here sign indicates the direction of crossings). Let us define Θ_i formally.

For each oriented edge $\overrightarrow{e}$ of G and each oriented edge $\overrightarrow{e^*}$ of $D(G)$, define
$$\Theta_{\overrightarrow{e}}(\overrightarrow{e^*})=\begin{cases}1 & \text{if } D(\overrightarrow{e})=\overrightarrow{e^*};\\ -1 & \text{if } D(\overrightarrow{e})=-\overrightarrow{e^*};\\ 0 & \text{otherwise.}\end{cases}$$

For every $\phi=\sum_i\lambda_i\overrightarrow{e_i}\in\mathcal{E}(G)$ and every $\sigma=\sum_j\mu_j\overrightarrow{e^*_j}\in\mathcal{E}(D(G))$, we define
$$\Theta_\phi(\sigma)=\sum_i\sum_j\lambda_i\mu_j\Theta_{\overrightarrow{e_i}}(\overrightarrow{e^*_j}),$$

which counts the directed number of times ϕ and π cross each other. In algebraic topology, $\Theta_\phi(\sigma)$ is referred to as a *cap product*. It is easy to check that the above is well defined. We write Θ_i as a shorthand for $\Theta_{\overrightarrow{c(i)}}$. Finally, define $\Theta : \mathcal{E}(D(G)) \to \mathbb{Z}^{2g}$ by setting

$$\Theta(\sigma) = \big(\Theta_1(\sigma), \ldots, \Theta_{2g}(\sigma)\big)$$

for every $\sigma \in \mathcal{E}(D(G))$.

We have the following proposition which effectively says that any cycle of $D(G)$ that intersects each $\overrightarrow{c(i)}$ a net number of zero times is a boundary. It is very much what we expect from the properties of homology.

Lemma 4. *The function Θ defined above is a surjective homomorphism from $\mathcal{C}(D(G))$ to $\mathbb{Z}^{2g}$ whose kernel is $\mathcal{B}(D(G))$.*

For convenience, we combine some of the results we have so far to give the following proposition.

Proposition 2. *There exists an element $\sigma \in \mathcal{B}(D(G))$ that minimizes $\hat{d}^f_{D(G)}$ and satisfies the following properties.*

(a) We can write σ as $\overrightarrow{w_1} + \cdots + \overrightarrow{w_r}$ where the $\overrightarrow{w_i}$ are oriented circuits in $D(G)$ and $r \le g+1$.
(b) We have $|\sigma| = |\overrightarrow{w_1}| + \cdots + |\overrightarrow{w_r}|$ with $|\overrightarrow{w_i}| \le m$ for all i.
(c) $\Theta(\sigma) = \Theta(\overrightarrow{w_1}) + \cdots + \Theta(\overrightarrow{w_r}) = \mathbf{0}$.

Proof. Statements (a) and (b) follow from Lemma 3, and statement (c) follows from Lemma 4. □

Next we define a type of *covering graph* of $D(G)$ in which certain shortest paths will correspond to elements of $\mathcal{C}(D(G))$ whose sum will minimize $\hat{d}^f_{D(G)}$. The general construction we use is often referred to as the *derived graph of a voltage graph.*

We construct an infinite multigraph $H = (V_H, E_H)$ from $D(G) = (V', E')$ as follows. We set $V_H = V' \times \mathbb{Z} \times \mathbb{Z}^{2g}$. For convenience, we describe the oriented edges of H before describing its edges. For each oriented edge $\overrightarrow{e} = (u_1, e, u_2)$ of $D(G)$ and each $(k, \mathbf{v}) \in \mathbb{Z} \times \mathbb{Z}^{2g}$, we have an oriented edge of H, denoted $(\overrightarrow{e}, k, \mathbf{v})^*$, from

$$(u_1, k, \mathbf{v}) \quad \text{to} \quad (u_2, k + \hat{w}(\overrightarrow{e}), \mathbf{v} + \Theta(\overrightarrow{e})).$$

The same edge oriented oppositely is given by $(\overleftarrow{e}, k + \hat{w}(\overrightarrow{e}), \mathbf{v} + \Theta(\overrightarrow{e}))^*$. We write $(\overrightarrow{e}, k, \mathbf{v})$ for the edge of H corresponding to the oriented edge $(\overrightarrow{e}, k, \mathbf{v})^*$; thus every edge of H has two labels.

Walks of $D(G)$ naturally correspond to walks of H as follows. Let $w = (u_0, e_1, u_1, \ldots, e_{t-1}, u_t)$ be a walk of $D(G)$. Let $w_i = (u_0, e_1, u_1, \ldots, e_{i-1}, u_i)$ be the same walk up to the ith vertex, and let $\overrightarrow{e_i} = (u_{i-1}, e_i, u_i)$. Let $H(w)$ be the walk in H given by $H(w) = (u_0^H, e_1^H, u_1^H, \ldots, e_{t-1}^H, u_t^H)$, where $u_0^H = (u, 0, \mathbf{0})$ and

$$u_i^H = (u_i, \hat{w}(\overrightarrow{w_i}), \Theta(\overrightarrow{w_i})) \quad \text{and} \quad e_i^H = (\overrightarrow{e_i}, \hat{w}(\overrightarrow{w_{i-1}}), \Theta(\overrightarrow{w_{i-1}})).$$

It is easy to check that $w \mapsto H(w)$ is a bijective correspondence between walks of $D(G)$ and walks of H that start at $(u, 0, \mathbf{0})$ for some $u \in V'$. Furthermore, w

is a walk of $D(G)$ from u to u' and satisfies $\hat{w}(\overrightarrow{w}) = k$ and $\Theta(\overrightarrow{w}) = \mathbf{v}$ if and only if $H(w)$ is a walk of H from $(u, 0, \mathbf{0})$ to $(u', k, \mathbf{v})$.

Defining $V_H^* = V' \times \{-mn, \dots, mn\} \times \{-m, \dots, m\}^{2g} \subset V_H$, let H^* be the finite graph induced by H on V_H^*. For $(u, k, \mathbf{v}) \in V_H^*$, define $p(u, k, \mathbf{v})$ to be a shortest path in H^* from $(u, 0, \mathbf{0})$ to $(u, k, \mathbf{v})$ (if it exists); this path corresponds to a closed walk in $D(G)$. For fixed k and $\mathbf{v}$, let $p(k, \mathbf{v})$ be the path of minimum length in $\{p(u, k, \mathbf{v}) : u \in V'\}$ (if it exists). Let $w(k, \mathbf{v})$ be the (closed) walk of $D(G)$ corresponding to the path $p(k, \mathbf{v})$ in H^*. We have the following lemma.

Lemma 5. *Suppose w is a circuit of $D(G)$ such that $\hat{w}(\overrightarrow{w}) = k$ and $\Theta(\overrightarrow{w}) = \mathbf{v}$ and $|\overrightarrow{w}| \leq m$. Then $|\overrightarrow{w}| \geq |\overrightarrow{w(k, \mathbf{v})}|$.*

Let W be the set of all the $\overrightarrow{w(k, \mathbf{v})}$. Let $X = \{\overrightarrow{w_1} + \cdots + \overrightarrow{w_r} : \overrightarrow{w_i} \in W \,\forall i$ and $r \leq g + 1\}$, and let $Y = \{\sigma \in X : \Theta(\sigma) = \mathbf{0}\}$. We have the following corollary.

Corollary 1. *There exists an element of Y that minimizes $\hat{d}^f_{D(G)}$.*

We are now ready to present our algorithm and to prove its correctness.

3 The Algorithm

In this section, we present the basic steps of our algorithm and compute its running time. In order to keep the presentation simple, we do not optimize the running time. Our algorithm runs in time $O(n^{2g^2+4g+7})$.

Let $f : [0, \frac{1}{2}] \to [0, \infty)$ be a fixed concave, increasing function that is computable in polynomial time on the rationals, and let g be a fixed non-negative integer. The input for our algorithm is an n-vertex undirected graph G of genus g. Since $n = |V|$ then $m = |E| = O(n)$ from Euler's formula. Using the result of Mohar [12] mentioned earlier, we can find an embedding of G on a surface Σ of genus g in $O(n)$ time (where the constant factor strongly depends on g). From the embedding we can construct (in $O(n)$ time) the dual graph $D(G) = (V', E')$ together with the function D which maps each oriented edge $\overrightarrow{e} \in \overrightarrow{E}$ of G to its dual $D(\overrightarrow{e}) \in \overrightarrow{E'}$ in $D(G)$. We have $|E'| = |E| = O(n)$, and from Euler's formula, we have $|V'| = O(n)$. By the result of Erickson and Whittlesey [5] mentioned earlier, we can find a combinatorial system of loops $\overrightarrow{c(1)}, \dots, \overrightarrow{c(2g)}$ of G in time $O(n)$. At this point the algorithm no longer requires the embedding of G.

Next we construct and store the (restricted) functions $\hat{w} : \overrightarrow{E'} \to \mathbb{Z}$ and $\Theta : \overrightarrow{E'} \to \mathbb{Z}^{2g}$. The computation and storage of these functions imposes an insignificant time cost in the final analysis, so any crude bound on the running time is sufficient. It is easy to check that both functions can be computed and stored in time $O(n^2)$.

From the (restricted) functions $\hat{w}$ and Θ, we can construct the graph H^* directly from its definition (given in the previous section). Observe that H^* has $O(n) \cdot (2mn+1) \cdot (2m+1)^{2g} = O(n^{2g+3})$ vertices and $O(n^{2g+3})$ edges. Thus it takes $O(n^{2g+3})$ time to construct H^*. For each $(u, k, \mathbf{v}) \in V_{H^*}$, we compute and store the shortest path $p(u, k, \mathbf{v})$ from $(u, 0, \mathbf{0})$ to $(u, k, \mathbf{v})$ in H^*. Finding each shortest path requires $O(|V_{H^*}|\log(|V_{H^*}|)) = O(n^{2g+4})$ time using Dijkstra's algorithm, and so, computing all the $p(u, k, \mathbf{v})$ requires $|V_{H^*}|O(n^{2g+4}) = O(n^{4g+7})$ time. For each fixed $k \in \{-mn, \dots, mn\}$ and $\mathbf{v} \in \{-n, \dots, n\}^{2g}$, we compute and store $p(k, \mathbf{v})$, the path of minimum length amongst the $p(u, k, \mathbf{v})$, and we use $p(k, \mathbf{v})$ to compute and store $w(k, \mathbf{v})$ and $\overrightarrow{w(k, \mathbf{v})}$ (recall that $w(k, \mathbf{v})$ is the closed walk in $D(G)$ corresponding to $p(k, \mathbf{v})$ as described in the previous section). The time cost so far is $O(n^{4g+7})$.

Recall the sets W, X, and Y from the previous section. Having stored the set W of all walks $\overrightarrow{w(k, \mathbf{v})}$, we compute and store the set

$$X = \{\overrightarrow{w_1} + \cdots + \overrightarrow{w_r} : \overrightarrow{w_i} \in W \; \forall i \text{ and } r \leq g+1\}.$$

Adding elements of W together requires $O(n)$ time; hence computing and storing X requires $O(n|X|) = O(n|W|^{g+1}) = O(n \cdot n^{2(g+1)^2})$ time. We compute $\Theta(\sigma)$ for every $\sigma \in X$ and store the set $Y = \{\sigma \in X : \Theta(\sigma) = \mathbf{0}\}$, which takes $O(n|X|)$ time. Finally we find an element σ^* of Y that minimizes $\hat{d}^f_{D(G)}$, which takes $O(n|Y|)$ time, and from Corollary 1 we know σ^* minimizes $\hat{d}^f_{D(G)}$ over all elements of $\mathcal{B}(D(G)$. Thus $d^f(G)$ is given by $\hat{d}^f_{D(G)}(\sigma^*)$, and the total time taken to find σ^* is dominated by $\max(O(n^{2(g+1)^2+1}), O(n^{4g+7})) = O(n^{2g^2+4g+7})$.

In order to find an f-sparest cut of G, we compute $\phi^* = D^{-1}(\sigma^*)$ and decompose it into a sum of cuts to give

$$\phi = \sum_{i=1}^{k} \overrightarrow{[S_i, \bar{S_i}]},$$

where $k = O(mn)$ from the bound on the size of H^*. Now one of the cuts $[S_i, \bar{S_i}]$ is an f-sparsest cut by the proof of Lemma 2, and such a cut can be found in $O(n^2)$ time once σ^* has been found.

4 Open Problems

An obvious question that arises from this work is whether the running time of our algorithm can be improved. Specifically, it would be interesting to know if the problem of finding the edge expansion of a graph is fixed parameter tractable with respect to genus.

Our algorithm crucially relies on our graph being embedded on an orientable surface. In particular, we use the fact that a graph embedded on an orientable surface has a *directed* dual; this is not the case for graphs embedded on non-orientable surfaces. It would be interesting to develop methods for finding edge expansion of graphs embedded on non-orientable surfaces.

Acknowledgements

The author thanks the anonymous reviewer whose detailed comments and suggested references have improved the paper.

References

1. Ambuhl, C., Mastrolilli, M., Svensson, O.: Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. In: FOCS 2007: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, Washington, DC, USA, pp. 329–337. IEEE Computer Society, Los Alamitos (2007)
2. Arora, S., Rao, S., Vazirani, U.: Expander flows, geometric embeddings and graph partitioning. J. ACM 56(2), Art. 5, 37 (2009)
3. Bonsma, P.S.: Linear time algorithms for finding sparsest cuts in various graph classes. In: 6th Czech-Slovak International Symposium on Combinatorics, Graph Theory, Algorithms and Applications. Electron. Notes Discrete Math, vol. 28, pp. 265–272. Elsevier, Amsterdam (2007)
4. Chambers, E.W., Erickson, J., Nayyeri, A.: Minimum cuts and shortest homologous cycles. In: SCG 2009: Proceedings of the 25th annual symposium on Computational geometry, pp. 377–385. ACM, New York (2009)
5. Erickson, J., Whittlesey, K.: Greedy optimal homotopy and homology generators. In: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1038–1046. ACM, New York (2005) (electronic)
6. Garey, M.R., Johnson, D.S.: Computers and intractability. In: A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences. W. H. Freeman and Co, San Francisco (1979)
7. Giblin, P.J.: Graphs, surfaces and homology, 2nd edn. Chapman & Hall, London (1981); An introduction to algebraic topology, Chapman and Hall Mathematics Series
8. Gross, J.L., Tucker, T.W.: Topological graph theory. Dover Publications Inc., Mineola (2001); Reprint of the 1987 original [Wiley, New York; MR0898434 (88h:05034)] with a new preface and supplementary bibliography
9. Leighton, T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J. ACM 46(6), 787–832 (1999)
10. Matula, D.W., Shahrokhi, F.: Sparsest cuts and bottlenecks in graphs. Discrete Appl. Math. 27(1-2), 113–123 (1990); Computational algorithms, operations research and computer science, Burnaby, BC (1987)
11. Mohar, B.: Isoperimetric numbers of graphs. J. Combin. Theory Ser. B 47(3), 274–291 (1989)
12. Mohar, B.: A linear time algorithm for embedding graphs in an arbitrary surface. SIAM J. Discrete Math. 12(1), 6–26 (1999) (electronic)
13. Mohar, B., Thomassen, C.: Graphs on surfaces. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore (2001)
14. Park, J.K., Phillips, C.A.: Finding minimum-quotient cuts in planar graphs. In: STOC, pp. 766–775 (1993)
15. Rao, S.B.: Faster algorithms for finding small edge cuts in planar graphs. In: STOC 1992: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing, pp. 229–240. ACM, New York (1992)
16. Shmoys, D.M.: Approximation algorithms for NP-hard problems, ch. 5. PWS Publishing Co., Boston (1997)

Constructing the R* Consensus Tree of Two Trees in Subcubic Time

Jesper Jansson[1,⋆] and Wing-Kin Sung[2,3]

[1] Ochanomizu University, 2-1-1 Otsuka, Bunkyo-ku, Tokyo 112-8610, Japan
`Jesper.Jansson@ocha.ac.jp`

[2] School of Computing, National University of Singapore, 3 Science Drive 2, Singapore 117543
`ksung@comp.nus.edu.sg`

[3] Genome Institute of Singapore, 60 Biopolis Street, Genome, Singapore 138672

Abstract. The previously fastest algorithms for computing the R* consensus tree of two given (rooted) phylogenetic trees T_1 and T_2 with a leaf label set of cardinality n run in $\Theta(n^3)$ time [3,8]. In this manuscript, we describe a new $O(n^2 \log n)$-time algorithm to solve the problem. This is a significant improvement because the R* consensus tree is defined in terms of a set $\mathcal{R}_{maj}$ which may contain $\Omega(n^3)$ elements, so any direct approach which explicitly constructs $\mathcal{R}_{maj}$ requires $\Omega(n^3)$ time.

1 Introduction

Phylogenetic trees are leaf-labeled trees which are commonly used to describe the evolutionary history of a set of objects such as biological species or languages [5,6,10]. Typically, in a phylogenetic tree, each leaf represents one of the objects being studied and the branching structure of the tree indicates the assumed evolutionary relationships among the objects. A *consensus tree* is a single phylogenetic tree which summarizes the branching information contained in an input collection of phylogenetic trees with identical leaf label sets. Consensus trees are useful when different data sets or different tree inference methods have produced a set of trees with the same leaf label set and slightly conflicting structures, yet a single tree is required to represent all of them [5]. Also, by exclusion, a consensus tree indicates areas of conflict in the input trees [2]. Furthermore, consensus trees are sometimes used as a basis for new phylogenetic inferences [2].

Many types of consensus trees have been defined and studied in detail. For a survey, see, e.g., [2]. Different types of consensus trees use different criteria to resolve conflicts among the input trees, so their mathematical properties vary. Therefore, the most suitable type of consensus tree to use in practice depends on the particular application. In this paper, we consider the so-called *R* consensus tree*. One advantage of the R* consensus tree is that it provides a statistically consistent estimator of the species tree topology when combining a set of gene

⋆ Funded by the Special Coordination Funds for Promoting Science and Technology, Japan.

trees, as recently demonstrated by Degnan *et al.* [4]; moreover, R* consensus outperformed other methods such as majority-rule consensus in the study conducted by [4]. For the case of two input trees, it is known that the R* consensus tree is equivalent to the *RV-III tree* introduced in [8].

The R* consensus tree is defined formally in Section 2 below. In short, given a set of input trees on a leaf label set L, if the rooted triplet $xy|z$ for any $\{x, y, z\} \subseteq L$ is consistent with more input trees than each of the two rooted triplets $xz|y$ and $yz|x$ is, then $xy|z$ belongs to a set named $\mathcal{R}_{maj}$. The R* consensus tree is the tree τ having the largest possible number of internal nodes such that every rooted triplet consistent with τ belongs to $\mathcal{R}_{maj}$.

The goal of this paper is to develop a fast algorithm for constructing the R* consensus tree for two input trees T_1 and T_2 with a leaf label set L of cardinality n. The previous algorithms for this problem require $\Theta(n^3)$ time [3,8]. Our main result is to reduce the time complexity to subcubic. When T_1 and T_2 have similar branching structures, $|\mathcal{R}_{maj}| = \Omega(n^3)$. Hence, in order to obtain a fast algorithm, we have to avoid explicitly constructing the set $\mathcal{R}_{maj}$. For this purpose, we use an alternative formulation based on distances from leaves to lowest common ancestors of pairs of leaves to compute the Apresjan clusters of a function $s_{\mathcal{R}_{maj}}$ associated to $\mathcal{R}_{maj}$. Then, we find the strong clusters of $\mathcal{R}_{maj}$ which are subsequently used to build the R* consensus tree. The running time of our new algorithm `R*_consensus_tree` (described in Section 3) is $O(n^2 \log n)$.

2 Preliminaries

2.1 Basic Definitions

A *phylogenetic tree* is a rooted, unordered, distinctly leaf-labeled tree in which every internal node has at least two children. From here on, "tree" means "phylogenetic tree", and every leaf in a tree is identified with its label.

We shall use the following terminology and notation. For any tree T and any internal node u of T, the subtree of T rooted at u is denoted by $T[u]$. The set of all leaves in a tree T is written as $\Lambda(T)$. For any two nodes u and v in a tree T, the *lowest common ancestor of* u *and* v *in* T is denoted by $lca^T(u, v)$. A *triplet* is a tree with exactly three leaves. Any *non-binary* tree with exactly three leaves $\{x, y, z\}$ is called a *fan triplet* and is written as $x|y|z$. On the other hand, any *binary* tree with exactly three leaves $\{x, y, z\}$ is called a *rooted triplet*, and is denoted by $xy|z$ if the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of x and z. Note that there are precisely four different triplets for any set of three leaf labels $\{x, y, z\}$, namely $x|y|z$, $xy|z$, $xz|y$, and $yz|x$ (see Fig. 1).

For any tree T and $\{x, y, z\} \subseteq \Lambda(T)$, the fan triplet $x|y|z$ is said to be *consistent with* T if $lca^T(x, y) = lca^T(x, z) = lca^T(y, z)$. Similarly, the rooted triplet $xy|z$ is *consistent with* T if $lca^T(x, y)$ is a proper descendant of $lca^T(x, z) = lca^T(y, z)$. Let $T||_{\{x,y,z\}}$ denote the unique fan triplet or rooted triplet with leaf label set $\{x, y, z\}$ which is consistent with T. Finally, for any tree T, let $r(T)$ be the set of all rooted triplets which are consistent with T, i.e., define

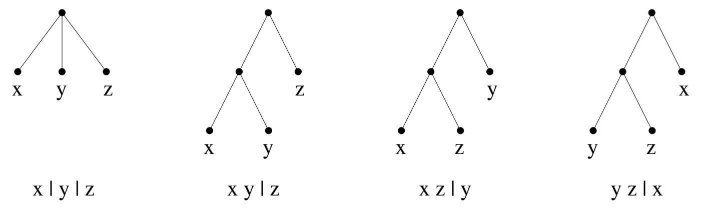


Fig. 1. The four different triplets $x|y|z$, $xy|z$, $xz|y$, and $yz|x$ leaf-labeled by $\{x, y, z\}$

$r(T) = \{T||_{\{x,y,z\}} : \{x, y, z\} \subseteq \Lambda(T)$ and $T||_{\{x,y,z\}}$ is not a fan triplet$\}$, and define $t(T)$ as the set of *all* triplets (rooted triplets as well as fan triplets) consistent with T, i.e., $t(T) = \{T||_{\{x,y,z\}} : \{x, y, z\} \subseteq \Lambda(T)\}$. It follows that $|t(T)| = \Theta(|\Lambda(T)|^3)$ for any tree T, and $|r(T)| = \Theta(|\Lambda(T)|^3)$ when T is a binary tree because $|r(T)| = |t(T)|$ in this case.

2.2 Strong Clusters and Apresjan Clusters

Below, let $\mathcal{R}$ be a given set of triplets over a leaf label set $L = \bigcup_{r \in \mathcal{R}} \Lambda(r)$ such that for each $\{x, y, z\} \subseteq L$, at most one of $x|y|z$, $xy|z$, $xz|y$, and $yz|x$ belongs to $\mathcal{R}$. A *cluster of* L is any subset of L. We define two special types of clusters:

- A cluster A of L is called a *strong cluster of* $\mathcal{R}$ if $aa'|x \in \mathcal{R}$ for all $a, a' \in A$ and $x \in L \setminus A$. Furthermore, L as well as every singleton set of L is also defined to be a strong cluster of $\mathcal{R}$.
- For each $a, b \in L$, define $s_{\mathcal{R}}(a, b) = |\{ab|y : ab|y \in \mathcal{R}\}|$. A cluster A of L is called an *Apresjan cluster of* $s_{\mathcal{R}}$ if $s_{\mathcal{R}}(a, a') > s_{\mathcal{R}}(a, x)$ for all $a, a' \in A$ and $x \in L \setminus A$.

Write $n = |L|$. By Theorem 2.3 and Corollary 2.1 of [3], the following holds:

Lemma 1. *(Bryant and Berry [3].)*

1. *There are $O(n)$ Apresjan clusters of $s_{\mathcal{R}}$.*
2. *Given the values of $s_{\mathcal{R}}(a, b)$ for all $a, b \in L$, the Apresjan clusters of $s_{\mathcal{R}}$ can be computed in $O(n^2)$ time.*

There is a relationship between the strong clusters of $\mathcal{R}$ and the Apresjan clusters of $s_{\mathcal{R}}$:

Lemma 2. *Every strong cluster of $\mathcal{R}$ is an Apresjan cluster of $s_{\mathcal{R}}$.*

Proof. Let C be a strong cluster of $\mathcal{R}$. Consider any fixed $a, a' \in C$ and $x \in L \setminus C$. For every $y \in L \setminus C$, we have $aa'|y \in \mathcal{R}$ by the definition of a strong cluster, which gives $s_{\mathcal{R}}(a, a') = |\{aa'|y : aa'|y \in \mathcal{R}\}| \geq |L \setminus C|$. In the same way, $ab|x \in \mathcal{R}$ holds for every $b \in C$, which (along with the requirement that for each $\{x, y, z\} \subseteq L$, at most one of $x|y|z$, $xy|z$, $xz|y$, and $yz|x$ belongs to $\mathcal{R}$) implies that $ax|b \notin \mathcal{R}$. Thus, $s_{\mathcal{R}}(a, x) = |\{ax|y : ax|y \in \mathcal{R}\}| \leq |(L \setminus C) \setminus \{x\}| < |L \setminus C|$. We have just shown that $s_{\mathcal{R}}(a, a') > s_{\mathcal{R}}(a, x)$, so C is an Apresjan cluster of $s_{\mathcal{R}}$. □

2.3 R* Consensus Trees

Let T_1 and T_2 be two given trees with $\Lambda(T_1) = \Lambda(T_2) = L$. For any $\{a, b, c\} \subseteq L$, define $\#ab|c$ as the number of trees T_i for which $ab|c \in r(T_i)$. The set of "majority rooted triplets" $\mathcal{R}_{maj}$ is defined as $\{ab|c : a, b, c \in L$ and $\#ab|c > \#ac|b,\ \#bc|a\}$. An *$R^*$ consensus tree of T_1 and T_2* is a tree τ with $\Lambda(\tau) = L$ which satisfies $r(\tau) \subseteq \mathcal{R}_{maj}$ and which maximizes the number of internal nodes.

The next two lemmas describe some useful properties of the strong clusters of $\mathcal{R}_{maj}$.

Lemma 3. *Let T be a tree with $\Lambda(T) = L$ and $r(T) \subseteq \mathcal{R}_{maj}$. For any node u of T, $\Lambda(T[u])$ is a strong cluster of $\mathcal{R}_{maj}$.*

Proof. If u is a leaf or the root of T then $\Lambda(T[u])$ is trivially a strong cluster of $\mathcal{R}_{maj}$. If u is an internal node then for any two $a, a' \in \Lambda(T[u])$ and any $x \notin \Lambda(T[u])$, the triplet $aa'|x$ belongs to $\mathcal{R}_{maj}$, so $\Lambda(T[u])$ is a strong cluster of $\mathcal{R}_{maj}$. □

Lemma 4. *There exists a tree τ such that the set* $\{\Lambda(\tau[u]) : u$ is a node in $\tau\}$ *equals the set of strong clusters of $\mathcal{R}_{maj}$.*

Proof. First observe that for any two strong clusters A and B of $\mathcal{R}$, either $A \subsetneq B$, $B \subsetneq A$, or $A \cap B = \emptyset$. To prove this claim, suppose for the sake of contradiction that there are $x, y, z \in L$ such that $x \in A \setminus B,\ y \in B \setminus A$, and $z \in A \cap B$. Since A is a strong cluster, $xz|y \in \mathcal{R}$. Since B is a strong cluster, $yz|x \in \mathcal{R}$. By the definition of $\mathcal{R}_{maj}$, we cannot have both of $xz|y$ and $yz|x$ in $\mathcal{R}_{maj}$. The claim follows. This implies that the strong clusters of $\mathcal{R}_{maj}$ form a hierarchy (laminar family) on L.

Next, if A and B are strong clusters of $\mathcal{R}_{maj}$ with $A \subsetneq B$ then there must exist some strong cluster C of $\mathcal{R}_{maj}$ such that $C \subsetneq B$ and $C \cap A = \emptyset$. (To see this, take any $c \in B \setminus A$ and recall that $\{c\}$ is by definition a strong cluster of $\mathcal{R}_{maj}$.)

By these two observations, there exists a tree τ leaf-labeled by L such that: (1) each strong cluster A of $\mathcal{R}_{maj}$ corresponds to a node in τ whose set of descendants is precisely A; (2) $A \subsetneq B$, where A and B are strong clusters of $\mathcal{R}_{maj}$, implies that the node corresponding to A is a proper descendant of the node corresponding to B; and (3) every internal node of τ has at least two children. Hence, the lemma follows. □

Say that a tree T *includes* a cluster A of L if T contains a node u such that $\Lambda(T[u]) = A$.

Theorem 1. *The R^* consensus tree of T_1 and T_2 exists and is unique. In particular, it includes every strong cluster of $\mathcal{R}_{maj}$ and no other clusters of L.*

Proof. By Lemma 4, there exists a (unique) tree τ that includes all the strong clusters of $\mathcal{R}_{maj}$ and does not include any other clusters. For any $aa'|x \in r(\tau)$, there exists a node u in τ such that $a, a' \in \Lambda(\tau[u])$ and $x \notin \Lambda(\tau[u])$, i.e., $a, a' \in A$

and $x \notin A$ for some strong cluster A of $\mathcal{R}_{maj}$, which implies that $aa'|x \in \mathcal{R}_{maj}$. Thus, $r(\tau) \subseteq \mathcal{R}_{maj}$.

To prove the optimality of τ, suppose that there exists a tree τ' which satisfies $r(\tau') \subseteq \mathcal{R}_{maj}$ and has more internal nodes than τ. By Lemma 3, the set of leaves in each rooted subtree of τ' forms a strong cluster of $\mathcal{R}_{maj}$. Since τ' has more internal nodes than τ, it follows that τ' includes some strong cluster of $\mathcal{R}_{maj}$ which τ does not include. This contradicts Lemma 4. Hence, τ is an R* consensus tree of T_1 and T_2. □

3 Constructing the R* Consensus Tree

Based on the discussion in Sections 2.2 and 2.3, we can construct the R* consensus tree of two given trees T_1 and T_2 as follows: First compute $s_{\mathcal{R}_{maj}}$ and all the Apresjan clusters of $s_{\mathcal{R}_{maj}}$.[1] Then, check each Apresjan cluster to see if it is a strong cluster of $\mathcal{R}_{maj}$ (by Lemma 2, the set of strong clusters of $\mathcal{R}_{maj}$ is a subset of the set of Apresjan clusters of $s_{\mathcal{R}_{maj}}$). Finally, build a tree which includes all the strong clusters of $\mathcal{R}_{maj}$ in accordance with Theorem 1. The algorithm is named `R*_consensus_tree` and is outlined in Fig. 2.

```
Algorithm R*_consensus_tree
Input: Two trees $T_1, T_2$ with $\Lambda(T_1) = \Lambda(T_2)$.
Output: The R* consensus tree of $T_1$ and $T_2$.
1: Define $L = \Lambda(T_1) = \Lambda(T_2)$ and compute $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$ as described
   in Section 4.
2: Compute the Apresjan clusters of $s_{\mathcal{R}_{maj}}$.
3: for each Apresjan cluster $A$ of $s_{\mathcal{R}_{maj}}$ do
4:    Determine if $A$ is a strong cluster of $\mathcal{R}_{maj}$ as described in Section 5.
5: end for
6: Construct the R* consensus tree using all the strong clusters of $\mathcal{R}_{maj}$.
```

Fig. 2. Algorithm `R*_consensus_tree`

We now analyze the time complexity of Algorithm `R*_consensus_tree`. In Section 4 below, we shall describe how to implement step 1 in $O(n^2 \log n)$ time. Next, in step 2, the Apresjan clusters of $s_{\mathcal{R}_{maj}}$ can be computed by running the algorithm of Bryant and Berry [3], which takes $O(n^2)$ time according to Lemma 1 (see also Corollary 2.1 in [3]). There are $O(n)$ Apresjan clusters in the loop of step 3 by Lemma 1, and each one is checked in $O(n)$ time in step 4, as detailed in Section 5 below. Finally, the last step builds the R* consensus tree from the strong clusters of $\mathcal{R}_{maj}$ in $O(n^2)$ time. In summary, we have the following theorem.

[1] Recall from Section 2.2 that for a set $\mathcal{R}$ of triplets over a leaf label set L, the function $s_{\mathcal{R}}$ is defined on each $a, b \in L$ by $s_{\mathcal{R}}(a, b) = |\{ab|y : ab|y \in \mathcal{R}\}|$.

Theorem 2. *Algorithm* `R*_consensus_tree` *constructs the R^* consensus tree of T_1 and T_2 in $O(n^2 \log n)$ time.*

The following two sections are devoted to implementing steps 1 and 4 of Algorithm `R*_consensus_tree` efficiently.

4 Computing $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$

From the definition of $\mathcal{R}_{maj}$, we have:

Lemma 5. *For any $a, b, c \in L$, $ab|c \in \mathcal{R}_{maj}$ if and only if either*

1. *$ab|c \in t(T_1) \cap t(T_2)$; or*
2. *$ab|c \in t(T_1)$ and $a|b|c \in t(T_2)$; or*
3. *$a|b|c \in t(T_1)$ and $ab|c \in t(T_2)$.*

We introduce the following three auxiliary functions:

$$\begin{cases} count_1(a,b) = |\{w \in L \setminus \{a,b\} \,:\, ab|w \in t(T_1) \cap t(T_2)\}| \\ count_2(a,b) = |\{w \in L \setminus \{a,b\} \,:\, ab|w \in t(T_1),\ a|b|w \in t(T_2)\}| \\ count_3(a,b) = |\{w \in L \setminus \{a,b\} \,:\, a|b|w \in t(T_1),\ ab|w \in t(T_2)\}| \end{cases}$$

Then, Lemma 5 immediately gives $s_{\mathcal{R}_{maj}}(a, b) = count_1(a, b) + count_2(a, b) + count_3(a, b)$ for any $a, b \in L$.

To compute $count_1$, $count_2$, and $count_3$, we could preprocess T_1 and T_2 in $O(n)$ time so that lowest common ancestor queries in T_1 and T_2 can be answered in $O(1)$ time [1,7]. Then, for any given $a, b \in L$, a brute-force solution would easily obtain all of $count_1(a, b)$, $count_2(a, b)$, and $count_3(a, b)$ in $O(n)$ time by checking $T_1||_{\{a,b,w\}}$ and $T_2||_{\{a,b,w\}}$ for each $w \in L \setminus \{a, b\}$. This approach would therefore compute $count_i(a, b)$ for all $a, b \in L$ and all $i = 1, 2, 3$ in $O(n^3)$ time. In the rest of this section, we show how to obtain $count_i(a, b)$ for all $a, b \in L$ and $i = 1, 2, 3$ more efficiently.

First, in Section 4.1, we reformulate the $count_i(a, b)$-functions in terms of distances from leaves to lowest common ancestors of leaves. All of these distances may be computed in $O(n^2)$ time. Then, based on this alternative formulation, for any fixed leaf label $a \in L$, Section 4.2 describes an $O(n \log n)$-time method for computing $count_1(a, b)$ for all $b \in L \setminus \{a\}$, and Section 4.3 describes an $O(n)$-time method for computing $count_2(a, b)$ for all $b \in L \setminus \{a\}$. (By symmetry, we can obtain $count_3(a, b)$ for all $b \in L \setminus \{a\}$ in $O(n)$ time with the same technique as in Section 4.3.) To summarize, after $O(n^2)$ time preprocessing, $O(n \log n)$ time is enough to compute $count_1(a, b)$, $count_2(a, b)$, and $count_3(a, b)$ for all $b \in L \setminus \{a\}$ for any fixed $a \in L$. Therefore, we only need $O(n^2 \log n)$ time in total to obtain $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$.

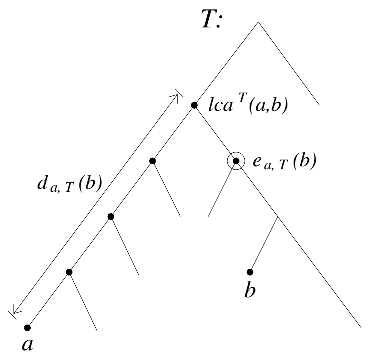


Fig. 3. Illustrating the definitions of $d_{a,T}(b)$ and $e_{a,T}(b)$

4.1 Distances from Leaves to Lowest Common Ancestors

For any tree T and any $a, b \in \Lambda(T)$, let $d_{a,T}(b)$ be the distance between a and $lca^T(a, b)$ in T, and let $e_{a,T}(b)$ be the child of $lca^T(a, b)$ which is an ancestor of b. (W.l.o.g., define $e_{a,T}(a) = \emptyset$.) See Fig. 3 for an example. Note that the values of $d_{a,T}(b)$ and $e_{a,T}(b)$ for all $a, b \in \Lambda(T)$ can be computed in $O(n^2)$ time by a bottom-up traversal. The next lemma states the connection between d and e and the triplets consistent with T.

Lemma 6. *For any tree T and any $a, x, w \in \Lambda(T)$, it holds that:*

- *If $d_{a,T}(x) < d_{a,T}(w)$ then $ax|w \in t(T)$.*
- *If $d_{a,T}(x) = d_{a,T}(w)$ and $e_{a,T}(x) \neq e_{a,T}(w)$ then $a|x|w \in t(T)$.*

Proof. If $d_{a,T}(x) < d_{a,T}(w)$, the $lca^T(a, w)$ is an ancestor of $lca^T(a, x)$. Thus, $ax|w \in t(T)$.

If $d_{a,T}(x) = d_{a,T}(w)$ then $v = lca^T(a, w) = lca^T(a, x)$. Since $e_{a,T}(x) \neq e_{a,T}(w)$, we know that $lca^T(x, w) = v$. Hence, $a|x|w \in t(T)$. □

We remark that the first part of Lemma 6 is a special case of Theorem 1 in [9], which was derived by Lee *et al.* [9] to solve a different problem known as *the maximum agreement subtree problem.*

Next, consider two trees T_1 and T_2 with $\Lambda(T_1) = \Lambda(T_2) = L$. We have:

Lemma 7. *For any $a, b, w \in L$:*

- *$count_1(a, b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w)$ and $d_{a,T_2}(b) < d_{a,T_2}(w)\}|$.*
- *$count_2(a, b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w)$, $d_{a,T_2}(b) = d_{a,T_2}(w)$, and $e_{a,T_2}(b) \neq e_{a,T_2}(w)\}|$.*
- *$count_3(a, b) = |\{w : d_{a,T_1}(b) = d_{a,T_1}(w)$, $e_{a,T_1}(b) \neq e_{a,T_1}(w)$, and $d_{a,T_2}(b) < d_{a,T_2}(w)\}|$.*

Proof. By Lemma 6, $d_{a,T_1}(b) < d_{a,T_1}(w)$ and $d_{a,T_2}(b) < d_{a,T_2}(w)$ mean that $ab|w \in t(T_1) \cap t(T_2)$. Hence, $count_1(a,b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w)$ and $d_{a,T_2}(b) < d_{a,T_2}(w)\}|$.

By Lemma 6 again, $d_{a,T_1}(b) < d_{a,T_1}(w)$, $d_{a,T_2}(b) = d_{a,T_2}(w)$, and $e_{a,T_2}(b) \neq e_{a,T_2}(w)$ mean that $ab|w \in t(T_1)$ and $a|b|w \in t(T_2)$. Hence, $count_2(a,b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w)$, $d_{a,T_2}(b) = d_{a,T_2}(w)$, and $e_{a,T_2}(b) \neq e_{a,T_2}(w)\}|$.

The formula for $count_3(a,b)$ follows by symmetry. □

4.2 Computing $count_1(a,b)$ for All $b \in L \setminus \{a\}$

This section describes how to compute $count_1(a,b)$ for all $b \in L \setminus \{a\}$ in $O(n \log n)$ time for any fixed $a \in L$, assuming the values of $d_{a,T_1}(b)$, $d_{a,T_2}(b)$ for all $b \in L \setminus \{a\}$ have been precomputed.

By applying a stable sorting, all elements in $L \setminus \{a\}$ can be ordered using $O(n)$ time so that x is before y if either: (1) $d_{a,T_1}(x) < d_{a,T_1}(y)$; or (2) $d_{a,T_1}(x) = d_{a,T_1}(y)$ and $d_{a,T_2}(x) < d_{a,T_2}(y)$. Suppose the resulting ordering of $L \setminus \{a\}$ is $x_{(1)}, x_{(2)}, \ldots, x_{(n-1)}$. For any $x_{(i)}$, let i' be the smallest integer which is larger than i such that $d_{a,T_1}(x_{(i')}) \neq d_{a,T_1}(x_{(i)})$. Then, $count_1$ obeys the following:

Lemma 8. *For any $i \in \{1, 2, \ldots, n-1\}$, $count_1(a, x_{(i)})$ equals the number of elements in $\{d_{a,T_2}(x_{(j)}) : j \geq i'\}$ which are strictly larger than $d_{a,T_2}(x_{(i)})$.*

Proof. From Lemma 7, $count_1(a, x_{(i)}) = |\{x_{(j)} : d_{a,T_1}(x_{(i)}) < d_{a,T_1}(x_{(j)})$ and $d_{a,T_2}(x_{(i)}) < d_{a,T_2}(x_{(j)})\}|$. By the definition of i', $count_1(a, x_{(i)}) = |\{x_{(j)} : j \geq i'$ and $d_{a,T_2}(x_{(i)}) < d_{a,T_2}(x_{(j)})\}|$. The lemma follows. □

We compute $count_1(a, x_{(i)})$ for all i in decreasing order from $n-1$ to 1 iteratively, as illustrated in Algorithm `Compute_`$count_1$ in Fig. 4. We maintain the invariant

```
Algorithm Compute_$count_1$
Input: $a \in L$.
Output: The values of $count_1(a,b)$ for all $b \in L \setminus \{a\}$.
 1: Perform a stable sorting to rank the elements of $L \setminus \{a\}$ according to $d_{a,T_1}$
    and $d_{a,T_2}$, and obtain the ordering $x_{(1)}, x_{(2)}, \ldots, x_{(n-1)}$.
 2: Let $D$ be an empty binary search tree.
 3: Let $i' = n$ and define $d_{a,T_1}(x_{(n)}) = d_{a,T_2}(x_{(n)}) = \infty$.
 4: for $i = n-1$ downto 1 do
 5:    if $d_{a,T_1}(x_{(i)}) < d_{a,T_1}(x_{(i+1)})$ then
 6:       Insert $d_{a,T_2}(x_{(j)})$ into $D$ for $j = i+1, \ldots, i'-1$.
 7:       Let $i' = i+1$.
 8:    end if
 9:    Set $count_1(a, x_{(i)})$ to the number of elements in $D$ which are strictly greater
       than $d_{a,T_2}(x_{(i)})$.
10: end for
```

Fig. 4. Algorithm `Compute_`$count_1$

that in every iteration i, the value i' equals the smallest integer larger than i such that $d_{a,T_1}(x_{(i')}) \neq d_{a,T_1}(x_{(i)})$, and keep a binary search tree D for $\{d_{a,T_2}(x_{(j)}) : j \geq i'\}$ so that by Lemma 8, $count_1(a, x_{(i)})$ equals the number of elements in D strictly greater than $d_{a,T_2}(x_{(i)})$. Each operation on a binary search tree takes $O(\log n)$ time, so in total, `Compute`_$count_1$_ runs in $O(n \log n)$ time.

4.3 Computing $count_2(a, b)$ for All $b \in L \setminus \{a\}$

Here, we show how to compute $count_2(a, b)$ (and by symmetry, $count_3(a, b)$) for all $b \in L \setminus \{a\}$ in $O(n)$ time for any fixed $a \in L$. We assume that the values of $d_{a,T_1}(b)$, $d_{a,T_2}(b)$, $e_{a,T_1}(b)$, $e_{a,T_2}(b)$ for all $b \in L \setminus \{a\}$ have been precomputed.

The algorithm is named `Compute`_$count_2$_ and is listed in Fig. 5. It first partitions $L \setminus \{a\}$ into disjoint subsets $C_1, C_2, \ldots, C_p$ in such a way that for every pair of elements x, y in the same C_i, it holds that $d_{a,T_2}(x) = d_{a,T_2}(y)$. Then, the algorithm further partitions each C_i into $C_{i1}, \ldots, C_{iq}$ so that for every pair of elements x, y in the same C_{ij}, it holds that $e_{a,T_2}(x) = e_{a,T_2}(y)$. Finally, the algorithm obtains $count_2(a, x)$ for all $x \in L \setminus \{a\}$ in $O(n)$ time based on Lemma 9.

Lemma 9. *Let $b \in L \setminus \{a\}$ and suppose $b \in C_{ij}$. Then, $count_2(a, b) = |\{w \in C_i : d_{a,T_1}(w) > d_{a,T_1}(b)\}| - |\{w \in C_{ij} : d_{a,T_1}(w) > d_{a,T_1}(b)\}|$.*

Proof. By Lemma 7, $count_2(a, b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w),\ d_{a,T_2}(b) = d_{a,T_2}(w)$, and $e_{a,T_2}(b) \neq e_{a,T_2}(w)\}|$. Since $b \in C_{ij}$, we have: (1) $w \in C_i$ means $d_{a,T_2}(b) = d_{a,T_2}(w)$; and (2) $w \in C_{ij}$ means $e_{a,T_2}(b) = e_{a,T_2}(w)$. Therefore, $count_2(a, b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w),\ w \in C_i$, and $w \notin C_{ij}\}|$. □

5 Determining If a Cluster Is a Strong Cluster

This section shows how to check whether a given cluster A of L is a strong cluster of $\mathcal{R}_{maj}$ in $O(n)$ time. Our solution depends on the following lemma.

Lemma 10. *Let u_1 and u_2 be the lowest common ancestor of A in T_1 and T_2, respectively. A is a strong cluster of $\mathcal{R}_{maj}$ if and only if:*

(1) For $i = 1, 2$, each subtree B attached to u_i in T_i satisfies either $\Lambda(B) \subseteq A$ or $\Lambda(B) \subseteq L \setminus A$; and
(2) $X_1 \cap X_2 = \emptyset$, where $X_i = (L \setminus A) \cap \Lambda(T_i[u_i])$.

Proof. ($\Rightarrow$) Let A be a strong cluster of $\mathcal{R}_{maj}$. We need to prove that both conditions (1) and (2) hold.

For the sake of obtaining a contradiction, suppose that for some $i \in \{1, 2\}$, there exist $a \in A$, $x \in L \setminus A$ where both a and x are in the same subtree attached to u_i. Then, $lca^{T_i}(a, x)$ is a proper descendant of u_i. The node u_i is defined as the lowest common ancestor of the leaves in A, so $lca^{T_i}(a, a') = u_i$ for some $a' \in A$. However, then $lca^{T_i}(a, x)$ is a proper descendant of $lca^{T_i}(a, a')$, giving $ax|a' \in t(T_i)$. This implies that $aa'|x$ cannot belong to $\mathcal{R}_{maj}$, which contradicts the fact that A is a strong cluster of $\mathcal{R}_{maj}$. Thus, condition (1) must hold.

```
Algorithm Compute_$count_2$
Input: $a \in L$.
Output: The values of $count_2(a, b)$ for all $b \in L \setminus \{a\}$.
 1: Partition $L \setminus \{a\}$ into $C_1, \ldots, C_p$ so that for every pair of elements $x, y$ in the
    same $C_i$, it holds that $d_{a,T_2}(x) = d_{a,T_2}(y)$.
 2: for every $C_i$ do
 3:    Perform a stable sorting to rank the elements of $C_i$ and obtain the ordering
       $b_1, \ldots, b_{|C_i|}$ such that $d_{a,T_1}(b_1) \leq d_{a,T_2}(b_2) \leq \ldots \leq d_{a,T_2}(b_{|C_i|})$.
 4:    { Note that $s_i(b) = |\{w \in C_i : d_{a,T_1}(w) > d_{a,T_1}(b)\}|$. }
 5:    $s_i(b_1) = 0$.
 6:    for $j = 2, 3, \ldots, |C_i|$ do
 7:       if $d_{a,T_1}(b_j) > d_{a,T_1}(b_{j-1})$ then
 8:          $s_i(b_j) = s_i(b_{j-1}) + 1$.
 9:       else
10:          $s_i(b_j) = s_i(b_{j-1})$.
11:       end if
12:    end for
13:    Partition $C_i$ into $C_{i1}, \ldots, C_{iq}$ so that for every pair of elements $x, y$ in the
       same $C_{ij}$, it holds that $e_{a,T_2}(x) = e_{a,T_2}(y)$.
14:    for every $C_{ij}$ do
15:       Perform a stable sorting to rank the elements of $C_{ij}$ and obtain the or-
          dering $b_1, \ldots, b_{|C_{ij}|}$ such that $d_{a,T_1}(b_1) \leq d_{a,T_2}(b_2) \leq \ldots \leq d_{a,T_2}(b_{|C_{ij}|})$.

16:       { Note that $s_{ij}(b) = |\{w \in C_{ij} : d_{a,T_1}(w) > d_{a,T_1}(b)\}|$.}
17:       $s_{ij}(b_1) = 0$.
18:       for $k = 2, 3, \ldots, |C_{ij}|$ do
19:          if $d_{a,T_1}(b_k) > d_{a,T_1}(b_{k-1})$ then
20:             $s_{ij}(b_k) = s_{ij}(b_{k-1}) + 1$.
21:          else
22:             $s_{ij}(b_k) = s_{ij}(b_{k-1})$.
23:          end if
24:       end for
25:       for every $b \in C_{ij}$ do
26:          Let $count_2(a, b)$ equal $s_i(b) - s_{ij}(b)$.
27:       end for
28:    end for
29: end for
```

Fig. 5. Algorithm Compute_$count_2$

Next, we prove by contradiction that condition (2) must hold. If $X_1 \cap X_2 \neq \emptyset$, where $X_i = (L \setminus A) \cap \Lambda(T_i[u_i])$, is true then there exists an $x \in X_1 \cap X_2$. We claim that there exist $a, a' \in A$ with $lca^{T_1}(a, a') = u_1$ and $lca^{T_2}(a, a') = u_2$. This yields $x|a|a' \in t(T_1)$ and $x|a|a' \in t(T_2)$ because the subtree attached to u_i for $i = 1, 2$ which contains x cannot contain a or a' by condition (1) above. Thus, $aa'|x \notin \mathcal{R}_{maj}$, contradicting the fact that A is a strong cluster of $\mathcal{R}_{maj}$.

It remains to prove the claim that there exist $a, a' \in A$ such that $lca^{T_1}(a, a') = u_1$ and $lca^{T_2}(a, a') = u_2$. Assume on the contrary that for each pair $a, a' \in A$,

```
Algorithm Check_if_strong_cluster
Input: A cluster $A$ of $L$.
Output: "Yes", if $A$ is a strong cluster of $\mathcal{R}_{maj}$; "no", otherwise.

 1: Let $u_1$ and $u_2$ be the lowest common ancestor of $A$ in $T_1$ and $T_2$, respectively.
 2: for $i = 1, 2$ do
 3:     if there exists a subtree in $T_i$ attached to $u_i$ containing leaves from $A$ as well
        as from $L \setminus A$ then
 4:         return "no"
 5:     end if
 6: end for
 7: Let $X_i = (L \setminus A) \cap \Lambda(T_i[u_i])$ for $i = 1, 2$.
 8: if $X_1 \cap X_2 = \emptyset$ then
 9:     return "yes"
10: else
11:     return "no"
12: end if
```

Fig. 6. Algorithm `Check_if_strong_cluster`

there exists an $i \in \{1, 2\}$ such that $lca^{T_i}(a, a')$ is a proper descendant of u_i. Note that A is located in at least two subtrees attached to u_i for $i = 1, 2$. For every pair of subtrees attached to u_1 that contain elements from A, suppose the sets of leaves in the two subtrees are A^1 and A^2. To ensure that $lca^{T_2}(x, y)$ is a proper descendant of u_2 for every $x \in A^1$ and $y \in A^2$, we need $A^1 \cup A^2$ to appear in one subtree attached to u_2. By this argument, all elements of A appear in one subtree attached to u_2. Then, u_2 cannot be the lowest common ancestor of A in T_2, and we arrive at a contradiction.

($\Leftarrow$) Suppose A satisfies the two conditions stated in the lemma. We need to prove that for every $x, y \in A$ and $z \in L \setminus A$, it holds that $xy|z \in \mathcal{R}_{maj}$. Note that z belongs to either X_1, X_2, or $((L \setminus A) \setminus X_1) \setminus X_2$, which gives three cases:

– If $z \in ((L \setminus A) \setminus X_1) \setminus X_2$: Then $lca^{T_i}(x, z)$ is a proper ancestor of $lca^{T_i}(x, y)$ for $i = 1, 2$. Thus, $xy|z \in \mathcal{R}_{maj}$.
– If $z \in X_1$: Then $xy|z \in t(T_2)$ and there are two cases depending on whether or not $lca^{T_1}(x, y)$ is a proper descendant of u_1. If yes, then $xy|z \in t(T_1)$, and thus $xy|z \in \mathcal{R}_{maj}$. If no, then $lca^{T_1}(x, y) = u_1$, and thus $x|y|z \in t(T_1)$, which also yields $xy|z \in \mathcal{R}_{maj}$.
– If $z \in X_2$: Then it follows that $xy|z \in \mathcal{R}_{maj}$ in the same way as for the case $z \in X_1$ above. □

We can use Lemma 10 to check if a given cluster A of L is a strong cluster of $\mathcal{R}_{maj}$ in $O(n)$ time, as shown in Algorithm `Check_if_strong_cluster` in Fig. 6.

6 Concluding Remarks

In this paper, we have described how to compute the R* consensus tree of two input trees in $O(n^2 \log n)$ time. The definition of the set of majority rooted triplets $\mathcal{R}_{maj}$ as well as the definition of an R* consensus tree can be naturally extended to the case of $k > 2$ input trees; see [2]. The currently fastest algorithm for the case $k > 2$ runs in $O(kn^3)$ time and is outlined in [2]. It can be implemented by explicitly constructing the sets $r(T_i)$ for every input tree T_i using $O(kn^3)$ total time, then obtaining $\mathcal{R}_{maj}$ in $O(kn^3)$ time by finding the most frequently occurring rooted triplet (if it exists) for each $\{x, y, z\}$ in the leaf label set in $O(k)$ time, and finally applying the $O(n^3)$-time strong clustering algorithm from Corollary 2.2 in [3] to $\mathcal{R}_{maj}$. An open problem is to reduce the time complexity to $o(kn^3)$. It seems difficult to extend the approach used in this paper because it would yield an exponential number (in k) of cases in Lemma 5 and thus express $s_{\mathcal{R}_{maj}}(a, b)$ as the sum of an exponential number of auxiliary *count*-functions, causing the total running time to be exponential in k.

Acknowledgments

The authors would like to thank David Bryant for some clarifications and the anonymous referees for their helpful comments.

References

1. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
2. Bryant, D.: A classification of consensus methods for phylogenetics. In: Janowitz, M.F., Lapointe, F.-J., McMorris, F.R., Mirkin, B., Roberts, F.S. (eds.) Bioconsensus. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 61, pp. 163–184. American Mathematical Society, Providence (2003)
3. Bryant, D., Berry, V.: A structured family of clustering and tree construction methods. Advances in Applied Mathematics 27(4), 705–732 (2001)
4. Degnan, J.H., DeGiorgio, M., Bryant, D., Rosenberg, N.A.: Properties of consensus methods for inferring species trees from gene trees. Systematic Biology 58(1), 35–54 (2009)
5. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Inc., Sunderland (2004)
6. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, New York (1997)
7. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM Journal on Computing 13(2), 338–355 (1984)
8. Kannan, S., Warnow, T., Yooseph, S.: Computing the local consensus of trees. SIAM Journal on Computing 27(6), 1695–1724 (1998)
9. Lee, C.-M., Hung, L.-J., Chang, M.-S., Shen, C.-B., Tang, C.-Y.: An improved algorithm for the maximum agreement subtree problem. Information Processing Letters 94(5), 211–216 (2005)
10. Nakhleh, L., Warnow, T., Ringe, D., Evans, S.N.: A comparison of phylogenetic reconstruction methods on an Indo-European dataset. Transactions of the Philological Society 103(2), 171–192 (2005)

Author Index

Abraham, Ittai II-87
Adler, Isolde I-97
Agarwal, Pankaj K. I-487
Ajwani, Deepak II-75
Anshelevich, Elliot I-158
Arya, Sunil I-374
Azar, Yossi II-51

Bae, Sang Won I-500
Bahmani, Bahman I-170
Bansal, Nikhil II-218
Bartal, Yair II-87
Bar-Yehuda, Reuven I-255
Bast, Hannah I-290
Belazzougui, Djamal I-427
Bendich, Paul I-1
Berenbrink, Petra I-134
Bhawalkar, Kshipra II-17
Boldi, Paolo I-427
Bonifaci, Vincenzo II-230
Brodal, Gerth Stølting II-171
Buchbinder, Niv II-51
Buchin, Kevin I-110, I-463, II-63
Buchin, Maike I-463, II-63
Büsing, Christina I-350

Carlsson, Erik I-290
Chan, Sze-Hang I-23
Chao, Kun-Mao I-415
Chechik, Shiri I-84
Chen, Kuan-Yu I-415
Chen, Ning II-147
Chrobak, Marek I-195
Cicalese, Ferdinando I-207
Colin de Verdière, Éric II-100
Cormode, Graham I-231
Crescenzi, Pierluigi I-302
Culpepper, J. Shane II-194
Cygan, Marek I-72
Czumaj, Artur I-410

da Fonseca, Guilherme D. I-374
Das, Abhimanyu I-219
Davoodi, Pooya II-171
Dickerson, Matthew T. I-362
Dorn, Frederic I-97

Edelsbrunner, Herbert I-1
Eigenwillig, Arno I-290
Eisenbrand, Friedrich I-11
Elkin, Michael I-48
Elmasry, Amr II-183
Elsässer, Robert I-134
Eppstein, David I-362

Feldman, Jon I-182
Ferragina, Paolo II-1
Fomin, Fedor V. I-97
Friedler, Sorelle A. I-386
Fujita, Ryo II-123
Fujiwara, Hiroshi I-439

Gairing, Martin II-17
Galli, Laura I-338
Gaspers, Serge I-267
Geisberger, Robert I-290
Ghosh, Arpita II-147
Gibson, Matt I-243
Giesen, Joachim I-524
Goodrich, Michael T. I-362
Grandoni, Fabrizio I-536
Grossi, Roberto I-302
Gupta, Anupam II-218
Gutin, Gregory I-326

Hajiaghayi, MohammadTaghi I-314
Halperin, Dan I-398
Harks, Tobias II-29
Harrelson, Chris I-290
Hellweg, Frank I-60
Hemmer, Michael I-398
Henzinger, Monika I-182
Hermelin, Danny I-255
Hoefer, Martin I-158, II-29

Iersel, Leo van I-326
Imbrenda, Claudio I-302
Iwama, Kazuo II-135

Jacobs, Tobias I-439
Jaggi, Martin I-524
Jain, Kamal II-51
Jansson, Jesper I-573

Kamiński, Marcin I-122
Kang, Ross J. II-112
Kaplan, Haim I-475
Kapralov, Michael I-170
Kärkkäinen, Juha I-451
Katz, Matthew J. I-475
Kempe, David I-219
Kesavan, Karthikeyan I-11
Khandekar, Rohit I-314
Klimm, Max II-29
Kobayashi, Yusuke II-123
Korman, Matias I-500
Kortsarz, Guy I-314
Korula, Nitish I-182
Kowalik, Lukasz I-72
Kreveld, Marc van I-463
Krysta, Piotr II-39

Lampis, Michael I-549
Lam, Tak-Wah I-23
Langberg, Michael I-84
Lanzi, Leonardo I-302
Laue, Sören I-524
Lee, Lap-Kei I-23
Li, Jian II-218
Löffler, Maarten I-463
Lorenz, Ulf I-512

Makino, Kazuhisa I-195, II-123
Marchetti-Spaccamela, Alberto II-230
Marino, Andrea I-302
Martin, Alexander I-512
Mattikalli, Raju S. I-11
Maue, Jens I-350
Mestre, Julián II-218
Mirrokni, Vahab S. I-182
Mitzenmacher, Michael I-231
Miyazaki, Shuichi II-135
Mnich, Matthias I-267, I-326, II-112
Morgenstern, Gila I-475
Morozov, Dmitriy I-1
Mount, David M. I-374, I-386
Mozes, Shay II-206
Mucha, Marcin I-72
Müller, Tobias II-112

Nagarajan, Viswanath II-218
Navarro, Gonzalo II-194
Neiman, Ofer II-87
Niemeier, Martin I-11
Nordsieck, Arnold W. I-11

Okamoto, Yoshio I-500
Osipov, Vitaly I-278

Pagh, Rasmus I-427
Patel, Amit I-1
Patel, Viresh I-561
Paulusma, Daniël I-122
Peleg, David I-84
Phillips, Jeff M. I-487
Pilipczuk, Marcin I-72
Pirwani, Imran A. I-243
Pountourakis, Emmanouil I-146
Puglisi, Simon J. I-451, II-194

Radhakrishnan, Jaikumar II-159
Rao, S. Srinivasa II-171
Rawitz, Dror I-255
Raychev, Veselin I-290
Roditty, Liam I-84
Roughgarden, Tim II-17
Rudra, Atri II-218

Sanders, Peter I-278
Sankowski, Piotr I-72
Sauerwald, Thomas I-134
Sau, Ignasi I-97
Schmidt, Melanie I-60
Schulman, Leonard J. II-87
Schulz, André I-110, II-63
Setter, Ophir I-398
Shah, Smit II-159
Shannigrahi, Saswata II-159
Sharir, Micha I-475
Silveira, Rodrigo I. I-463
Sitchinava, Nodari II-75
Skopalik, Alexander II-29
Skutella, Martin I-11, I-36
Sohler, Christian I-60
Solomon, Shay I-48
Stein, Cliff I-182
Stiller, Sebastian I-338
Sung, Wing-Kin I-573

Thaler, Justin I-231
Thilikos, Dimitrios M. I-97, I-122
Turpin, Andrew II-194

Vaccaro, Ugo I-207
Ventre, Carmine II-39
Verschae, José I-11, I-36
Vidali, Angelina I-146
Viger, Fabien I-290
Vigna, Sebastiano I-427

Wenk, Carola I-463
Wiese, Andreas I-11
Wiratma, Lionov I-463
Woeginger, Gerhard J. I-195
Wolf, Jan I-512
Wulff-Nilsen, Christian II-206

Xu, Haifeng I-195

Yanagisawa, Hiroki II-135
Yeo, Anders I-326
Yu, Hai I-487

Zeh, Norbert II-75
Zenklusen, Rico I-536