Two-dimensional Subset Selection for Hypervolume and Epsilon-Indicator

Karl Bringmann
Max Planck Institute for Informatics
Saarbrücken, Germany

Tobias Friedrich
Friedrich-Schiller-Universität Jena
Jena, Germany

Patrick Klitzke
Universität des Saarlandes
Saarbrücken, Germany

ABSTRACT

The goal of bi-objective optimization is to find a small set of good compromise solutions. A common problem for bi-objective evolutionary algorithms is the following subset selection problem (SSP): Given n solutions $P \subset \mathbb{R}^2$ in the objective space, select k solutions P^* from P that optimize an indicator function. In the hypervolume SSP we want to select k points P^* that maximize the hypervolume indicator $\mathcal{I}_{\text{hyp}}(P^*, r)$ for some reference point $r \in \mathbb{R}^2$. Similarly, the ε-indicator SSP aims at selecting k points P^* that minimize the ε-indicator $\mathcal{I}_{\text{eps}}(P^*, R)$ for some reference set $R \subset \mathbb{R}^2$ of size m (which can be $R = P$).

We first present a new algorithm for the hypervolume SSP with runtime $\mathcal{O}(n\,(k + \log n))$. Our second main result is a new algorithm for the ε-indicator SSP with runtime $\mathcal{O}(n \log n + m \log m)$. Both results improve the current state of the art runtimes by a factor of (nearly) n and make the problems tractable for new applications. Preliminary experiments confirm that the theoretical results translate into substantial empirical runtime improvements.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]:
Analysis of Algorithms and Problem Complexity

General Terms

Measurement, Archiving Algorithms, Performance, Hypervolume-Indicator, Epsilon-Indicator

1. INTRODUCTION

In the general Subset Selection Problem (SSP) we are given a set $P \subset \mathbb{R}^d$ of size n and a positive integer k. The task is to select a subset $P^* \subseteq P$ of size k that maximizes (or minimizes) $\mathcal{I}(P^*)$ for some indicator function $\mathcal{I}$. We study the two-dimensional case $d = 2$ and present new results for two common indicator functions.

The hypervolume indicator $\mathcal{I}_{\text{hyp}}(P^*, r)$ of a set P^* measures the volume of the space dominated by P^* up to some fixed reference point r (for formal definitions see Section 2). The aim of the *Hypervolume Subset Selection Problem* (HYPSSP) is to maximize $\mathcal{I}_{\text{hyp}}(P^*, r)$. The second indicator we consider is the ε-indicator $\mathcal{I}_{\text{eps}}(P^*, R)$, which is defined relative to a reference set $R \subset \mathbb{R}^d$ of size m (one may choose $P = R$) and measures how well P^* approximates R. The aim of the *ε-Indicator Subset Selection Problem* (EPSSSP) is to minimize $\mathcal{I}_{\text{eps}}(P^*, R)$.

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2662-9/14/07 ...$15.00.
http://dx.doi.org/10.1145/2576768.2598276.

Motivation. Subset selection problems occur frequently in all population-based search heuristics. We describe two specific applications. First, consider a hypervolume-based $(\mu + \lambda)$-evolutionary multiobjective algorithm (EMOA) such as SIBEA [20], SMS-EMOA [5], or the generational MO-CMA-ES [13, 14]. In every generation we have μ parents and generate λ offspring. From these $\mu + \lambda$ solutions we want to select the next population of size μ. Since our overall goal is to maximize the hypervolume indicator, we want to choose these μ points such that they maximize the hypervolume among all size-μ sets, which is an instance of HYPSSP by choosing $k = \mu$ and $n = \mu + \lambda$. Typically, $\lambda \approx \mu$ so that $k \approx n/2$. For a discussion why in general a greedy approach with $\lambda = 1$ cannot find a set with maximal hypervolume, see [7, 8].

Similarly, there are ε-indicator-based algorithms such as AGE [9], where we have a reference set R (that changes over time) and want to select the next generation such that we minimize the ε-indicator with respect to R.

A second motivation lies in the evaluation of indicator-based EMOAs by running them on test problems. When running many different EMOAs on a test problem, it would be nice to be able to compare them with the optimal set of μ points. Now, test problems are often designed in such a way that we know the Pareto front explicitly, but this does not directly yield the optimal hypervolume achievable with μ points. To compute this number, we may discretize the Pareto front, to get a good finite approximation P of the Pareto front. Solving HYPSSP on the point set P with $k = \mu$ now yields (an approximation of) the desired value, the optimal hypervolume achievable with μ points on this test problem. Note that in this situation we want to choose $n = |P|$ as large as possible to get a good approximation, while $k = \mu \ll n$.

Results on Hypervolume Subset Selection. For 2 dimensions there are algorithms which solve HYPSSP in time $\mathcal{O}(n^3)$ [4], $\mathcal{O}(kn^2)$ [3] and $\mathcal{O}(n^2)$ [16]. The only known lower bound is the trivial $\Omega(n)$. Note that the lower bounds of $\Omega(n \log n)$ for computing the hypervolume [6] and computing all contributions [12] in two dimensions do not imply a lower bound for HYPSSP as we are computing a set of points and not volumes. In higher dimensions, no algorithm is known that is faster than enumerating all $\binom{n}{k}$ subsets of P of size k (see, e.g., [8]). In Section 3 we present a novel algorithm that shows the following.

Theorem 1. *Two-dimensional* HYPSSP *can be solved in time*

$$\mathcal{O}(n\,(k + \log n)).$$

This improves all previous algorithms by roughly a factor n. Our preliminary experiments practically confirm this theoretical improvement (cf. [1]). Our algorithmic improvement is based on a rarely cited, but very useful idea of Brucker [10]. In programming contests this approach is known as the *convex hull trick* [2].

Results on ε-Indicator Subset Selection. EPSSSP was first studied by Ponte, Paquete, and Figueira [18], who gave an $\mathcal{O}(nm\log(nm))$ time algorithm for dimension $d = 2$. Recently, Vaz, Paquete, and Ponte [19] slightly improved this algorithm by lowering the log-factor, however, the runtime remains $\Omega(nm)$. Again, in higher dimensions, no algorithm is known that is faster than enumerating all subsets of P of size k.

In a different community, Koltun and Papadimitriou [15] studied a variant of EPSSSP (under the name of "approximately dominating representatives" of "skylines"). In their variant they do not fix a bound k on the size of the selected set $|P^*|$, but they fix a bound ε on the quality $\mathcal{I}_{\text{eps}}(P^*, R)$ and minimize $|P^*|$ with respect to this bound. Moreover, they only consider the case $P = R$. They showed that in 2 dimensions their variant can be solved in $\mathcal{O}(n)$ time (after sorting). Intuitively, their problem variant is simpler than ours, since any algorithm for our variant yields an algorithm for their variant (with roughly the same runtime) by using a binary search over $k \in \{1, \ldots, n\}$. In the opposite direction, such a reduction does not work, since the quantity $\varepsilon^* := \mathcal{I}_{\text{eps}}(P^*, R)$ is a real number. Hence binary search for ε^* may go on forever, computing better and better approximations of ε^*, but never the correct value. In this sense, we are tackling a harder problem in this paper.

We present in Section 4 a novel algorithm for EPSSSP in 2 dimensions. It is randomized in that its runtime is a random variable that is small in expectation and with high probability (i.e., with probability $1 - n^{-c}$ for any fixed $c > 0$). The result of our randomized algorithm is guaranteed to be correct. We use an extension of the result of Koltun and Papadimitriou [15] as a subroutine.

Theorem 2. *Two-dimensional* EPSSSP *can be solved with a randomized algorithm in time (in expectation and with high probability)*

$$\mathcal{O}(n \log n + m \log m).$$

This answers an open problem of Vaz et al. [19], of whether there is a subquadratic time algorithm for EPSSSP (although they might have intended a deterministic algorithm, while we present a randomized one). Again, this improves the state of the art by roughly a factor of n (at least if $n = m$), which is confirmed by preliminary experiments (cf. [1]).

2. PRELIMINARIES

We consider maximization problems with vector-valued objective functions $f\colon \mathcal{X} \to \mathbb{R}^d$, where $\mathcal{X}$ denotes an arbitrary search space. The feasible points $\mathcal{Y} := f(\mathcal{X})$ are called the objective space. In this paper we *only work in the objective space*, i.e., we always consider points in $\mathbb{R}^d$ and may ignore that these points stem from a search space $\mathcal{X}$.

We say that a point $p = (p_1, \ldots, p_d) \in \mathbb{R}^d$ *(weakly) dominates* a point $q = (q_1, \ldots, q_d) \in \mathbb{R}^d$ ($q \preceq p$) iff $p_i \geqslant q_i$ for all $1 \leqslant i \leqslant d$.

In the definitions in this Section 2 we consider the general case of $d \in \mathbb{N}, d \geqslant 2$. We note, however, that in all subsequent sections we will work in dimension $d = 2$.

2.1 Hypervolume indicator

For a set $P \subset \mathbb{R}^d$ the *hypervolume indicator* (with respect to the reference point $r \in \mathbb{R}^d$) is defined defined as

$$\mathcal{I}_{\text{hyp}}(P, r) := \int_{\mathbb{R}^d} A(z)\, dz,$$

where $A\colon \mathbb{R}^d \to \{0, 1\}$ is the *attainment function* and $A(z) = 1$ if and only if $r \preceq z$ and there is a point $p \in P$ with $z \preceq p$. Thus, $\mathcal{I}_{\text{hyp}}(P, r)$ measures the volume of the space "between" r and P.

We study the following problem:

Problem HYPSSP: Given $k \in \mathbb{N}$, $r \in \mathbb{R}^d$, and a set $P \subset \mathbb{R}^d$ of size n, compute a subset $P^* \subseteq P$ of size at most k that maximizes $\mathcal{I}_{\text{hyp}}(P^*, r)$.

We note three *standard simplifications*. First, we can without loss of generality assume that $r = (0, \ldots, 0)$, since translation of P by $-r$ reduces to this situation. We write

$$\mathcal{I}_{\text{hyp}}(P) := \mathcal{I}_{\text{hyp}}(P, (0, \ldots, 0))$$

for short. Then we can assume that $P \subset \mathbb{R}^d_{>0}$, since points with non-positive coordinates do not contribute to the hypervolume.

Second, we can assume that there are no two points $p, q \in P$ with $p \preceq q$ (in which case we say that P is *non-dominating*), since there is an optimal solution containing no dominated point: In any solution $P' \subseteq P$ that contains p we can replace p by q (or simply delete p, if P' already contains q) without increasing the size of P' or decreasing $\mathcal{I}_{\text{hyp}}(P')$. Thus, we may simply delete any dominated point in P. We note that in 2 dimensions we can delete all dominated points from P in $\mathcal{O}(n \log n)$ time [17], so this assumption can be ensured in our alloted runtime.

Finally, in 2 dimensions we can assume that the points $P = \{p_1, \ldots, p_n\}$ are *sorted* by increasing x-coordinate. It is not hard to see that, since P is non-dominating, in this case the points are also sorted by decreasing y-coordinate. Again this can be ensured in time $\mathcal{O}(n \log n)$.

In 2 dimensions, we denote the x- and y-coordinates of a point $p \in \mathbb{R}^2$ by p_x and p_y. Then for reference point $(0, 0)$ and for a sorted non-dominating set $P = \{p^1, \ldots, p^n\} \subset \mathbb{R}^d_{>0}$ we have $0 < p^1_x < \ldots < p^n_x$ and $p^1_y > \ldots > p^n_y > 0$.

2.2 ε-Indicator

In this paper we consider additive approximation. By taking the logarithm, our algorithm also works in the multiplicative setting. For points $p = (p_1, \ldots, p_d)$, $r = (r_1, \ldots, r_d) \in \mathbb{R}^d$, we set

$$\mathcal{I}_{\text{eps}}(p, r) := \max_{1 \leqslant i \leqslant d} r_i - p_i.$$

This denotes the minimal number ε by which we have to increase p in all coordinates so that it dominates q. This number measures how well p approximates q. For finite sets of points $P, R \subset \mathbb{R}^d$ the ε-indicator is defined as

$$\mathcal{I}_{\text{eps}}(P, R) := \max_{r \in R} \min_{p \in P} \mathcal{I}_{\text{eps}}(p, r).$$

This denotes the minimal number ε by which we have to increase all points in P in all coordinates so that every point in R is dominated by some point in P. It measures how well P approximates R. We say that P ε-approximates R if $\mathcal{I}_{\text{eps}}(P, R) \leqslant \varepsilon$.

In the following, we consider R as a fixed reference set and want to select a subset P^* of P that best approximates R subject to a size constraint. Note that it is allowed to set $P = R$.

Problem EPSSSP: Given $k \in \mathbb{N}$, $R \subset \mathbb{R}^d$ of size m, and a set $P \subset \mathbb{R}^d$ of size n, compute a subset $P^* \subseteq P$ of size at most k that minimizes $\mathcal{I}_{\text{eps}}(P^*, R)$.

Similar to the simplifications for HYPSSP, we may delete all dominated points in P (since there is an optimal solution containing no dominated point) and in R (since if we approximate the dominating point then we also approximate the dominated one). Additionally, we can assume that P and R are sorted. In 2 dimensions, this can again be ensured in the alloted runtime $\mathcal{O}(n \log n + m \log m)$. In the remainder of the paper we always work in $d = 2$ dimensions.

3. ALGORITHM FOR HYPSSP

After the standard simplifications described above in Section 2, we are given a set $P = \{p^1, \ldots, p^n\}$ with $0 < p_x^1 < \ldots < p_x^n$ and $p_y^1 > \ldots > p_y^n > 0$, and an integer k. Recall that our goal is to compute a set $P^* \subseteq P$ of size at most k that maximizes $\mathcal{I}_{\text{hyp}}(P^*) = \mathcal{I}_{\text{hyp}}(P^*, (0,0))$. We first present an algorithm that computes $\mathcal{I}_{\text{hyp}}(P^*)$, i.e., the maximal hypervolume instead of a set of points achieving the maximal hypervolume. In Section 3.3 we then explain how we can reconstruct P^* from our computation of $\mathcal{I}_{\text{hyp}}(P^*)$.

3.1 The Algorithm

Note that our reference point for $\mathcal{I}_{\text{hyp}}$ is $(0,0)$ (which we can assume by transforming the problem instance accordingly). During our computation we will, however, consider hypervolumes with respect to other reference points, as follows.

We define H_i^ℓ as the maximum hypervolume achievable with at most ℓ points and reference point $(p_x^i, 0)$, for any $0 \leqslant \ell \leqslant k$ and $1 \leqslant i \leqslant n$. Since the points $p^1, \ldots, p^i$ do not contribute anything when the reference point is $(p_x^i, 0)$, we have

$$H_i^\ell = \max_{Q \subseteq_\ell \{p^{i+1}, \ldots, p^n\}} \mathcal{I}_{\text{hyp}}(Q, (p_x^i, 0)),$$

where we use $A \subseteq_\ell B$ as a shorthand for $A \subseteq B$, $|A| \leqslant \ell$. We extend the definition of H_i^ℓ to $i = 0$ by setting $p_x^0 := 0$. Note that H_0^k is the maximal hypervolume of any k points in P with reference point $(0,0)$. In other words, $H_0^k = \mathcal{I}_{\text{hyp}}(P^*)$ is what we want to compute.

Moreover, we define $F_i^\ell(x)$ as the maximum hypervolume achievable with at most ℓ points, with p^i being the first of these points, and reference point $(x, 0)$, for any $1 \leqslant \ell \leqslant k$, $1 \leqslant i \leqslant n$, and $x \leqslant p_x^i$. Formally, we have

$$F_i^\ell(x) = \max_{\substack{Q \subseteq_\ell \{p^i, \ldots, p^n\} \\ p^i \in Q}} \mathcal{I}_{\text{hyp}}(Q, (x, 0))$$

We will use the functions $F_i^\ell(x)$ in order to compute the values $H_{i'}^{\ell'}$, and vice versa, on our way of computing H_0^k. The next lemma shows how to compute $H_0^\ell, \ldots, H_{n-1}^\ell$ from $F_1^\ell(x), \ldots, F_n^\ell(x)$.

Lemma 3. *For any $0 \leqslant i < n$ and $1 \leqslant \ell \leqslant k$ we have*

$$H_i^\ell = \max_{i < j \leqslant n} F_j^\ell(p_x^i).$$

Proof. If the selected set Q is empty then the hypervolume is $\mathcal{I}_{\text{hyp}}(Q, r) = 0$ (for any reference point r). Otherwise Q contains at least one point; let p^j be the leftmost of these points. We obtain

$$\begin{aligned} H_i^\ell &= \max_{Q \subseteq_\ell \{p^{i+1}, \ldots, p^n\}} \mathcal{I}_{\text{hyp}}(Q, (p_x^i, 0)) \\ &= \max\Big\{0, \max_{i < j \leqslant n} \max_{\substack{Q \subseteq_\ell \{p^j, \ldots, p^n\} \\ p^j \in Q}} \mathcal{I}_{\text{hyp}}(Q, (p_x^i, 0))\Big\} \\ &= \max\{0, \max_{i < j \leqslant n} F_j^\ell(p_x^i)\}. \end{aligned}$$

We finish the proof by noting that $F_j^\ell(p_x^i) \geqslant 0$. ☐

The second relation shows how to compute $F_i^\ell(x)$ from $H_i^{\ell-1}$.

Lemma 4. *For any $1 \leqslant i \leqslant n$, $1 \leqslant \ell \leqslant k$, and $x \leqslant p_x^i$ we have*

$$F_i^\ell(x) = -p_y^i \cdot x + \left(p_y^i p_x^i + H_i^{\ell-1}\right).$$

In particular, the above lemma shows that $F_i^\ell(x)$ is a linear function $a \cdot x + b$ in variable x. Its coefficients a and b can be read of Lemma 4 once we have computed $H_i^{\ell-1}$. Note that the slope of $F_i^\ell(x)$ is $\text{slope}(F_i^\ell(x)) = -p_y^i$ so that the functions $F_1^\ell, \ldots, F_n^\ell$ are sorted by increasing slope. Moreover, we can store the function $F_i^\ell(x)$ succinctly by only storing its coefficients a, b. This allows to evaluate F_i^ℓ at any given point x in constant time.

Proof of Lemma 4. Recall that in the definition of $F_i^\ell(x)$ the point p^i is required to be in the chosen set Q. We can split the hypervolume $F_i^\ell(x)$ into a part to the left of p_x^i (which is simply $\mathcal{I}_{\text{hyp}}(\{p^i\}, (x, 0))$) and a part to the right of p_x^i (which can be expressed as a hypervolume of at most $\ell - 1$ points with reference point $(p_x^i, 0)$, since we cut off at p_x^i). Formally, we have

$$\begin{aligned} F_i^\ell(x) &= \max_{\substack{Q \subseteq_\ell \{p^i, \ldots, p^n\} \\ p^i \in Q}} \mathcal{I}_{\text{hyp}}(Q, (x, 0)) \\ &= \mathcal{I}_{\text{hyp}}(\{p^i\}, (x, 0)) + \max_{Q \subseteq_{\ell-1} \{p^{i+1}, \ldots, p^n\}} \mathcal{I}_{\text{hyp}}(Q, (p_x^i, 0)). \end{aligned}$$

We finish the proof by noting that the first summand on the right hand side equals $p_y^i(p_x^i - x)$ and the second summand is $H_i^{\ell-1}$. ☐

Plugging these two lemmas together yields an algorithm for computing the desired maximal hypervolume $\mathcal{I}_{\text{hyp}}(P^*) = H_0^k$, see Algorithm 1. Observe that for initialization we can use $H_i^0 = H_n^\ell = 0$ for all $0 \leqslant i \leqslant n$ and $1 \leqslant \ell \leqslant k$.

Concerning runtime, note that the trivial evaluation of Lemma 3 (line 6 of Algorithm 1) takes $\Theta(n^2)$ time (as we have to iterate over $1 \leqslant i < j \leqslant n$). In the next section we present a faster evaluation method, Algorithm 2, that runs in time $\mathcal{O}(n)$. This speedup is the core trick of our new algorithm. It is easy to see that then Algorithm 1 runs in time $\mathcal{O}(nk)$. Since we have to ensure that the input P is non-dominating and sorted, which takes time $\mathcal{O}(n \log n)$, we obtain a total runtime of $\mathcal{O}(n\,(k + \log n))$ to solve HYPSSP. The space requirement is $\mathcal{O}(n)$, since we can forget $H_0^{\ell-1}, \ldots, H_n^{\ell-1}$ and $F_1^{\ell-1}, \ldots, F_n^{\ell-1}$ once we have computed $H_0^\ell, \ldots, H_n^\ell$ and $F_1^\ell, \ldots, F_n^\ell$.

3.2 Upper Envelope

We want to find a linear time algorithm for the following problem: Given a sequence of linear functions with increasing slopes and a set of x-coordinates given in increasing order, compute the maximal value of all those functions for each coordinate. More formally we are given $F_1^\ell(x), \ldots, F_n^\ell(x)$ with $\text{slope}(F_1^\ell) < \ldots < \text{slope}(F_n^\ell)$ and coordinates $p_x^0 < \ldots < p_x^{n-1}$ and want to compute the values

$$H_i^\ell = \max_{i < j \leqslant n} F_j^\ell(p_x^i) \quad \text{for all } 0 \leqslant i < n.$$

Algorithm 1: HYPSSP in 2 dimensions

```
  Input: set $P = \{p^1, .., p^n\}$ with
         $0 < p_x^1 < ... < p_x^n$ and $p_y^1 > ... > p_y^n > 0$
1 $p_x^0 := 0$
2 for $i = 0$ to $n$ do $H_i^0 := 0$
3 for $\ell = 1$ to $k$ do
4     for $i = 1$ to $n$ do compute (the coefficients of) the linear
      function $F_i^\ell(x) := -p_y^i \cdot x + \left(p_y^i p_x^i + H_i^{\ell-1}\right)$
5     $H_n^\ell := 0$
6     $H_i^\ell := \max_{i<j\leqslant n} F_j^\ell(p_x^i)$ for $0 \leqslant i < n$ (using Algorithm 2)
7 return $H_0^k$
```

Algorithm 2: Upper Envelope

```
  Input: coordinates $x_1 < .. < x_n$, linear functions $f_1, ..., f_n$
         with slope($f_1$) < ... < slope($f_n$)
1 Dequeue $S := \emptyset$
2 for $i = 1$ to $n$ do
3     while $|S| > 1$ and $\otimes(f_i, S_{|S|-1}) \leqslant \otimes(S_{|S|}, S_{|S|-1})$ do
4         $S$.popBack()   // delete $S_{|S|}$
5     $S$.append($f_i$)
6     while $|S| > 1$ and $\otimes(S_1, S_2) \leqslant x_i$ do
7         $S$.popFront()   // delete $S_1$
8     $h_i := S_1(x_i)$   // evaluate function $S_1$ at position $x_i$
9 return $h_1, \dots, h_n$
```

Note that the function $\max_{i<j\leqslant n} F_j^\ell(x)$ is an *upper envelope* of linear functions. Since the space above such an upper envelope is convex, the technique of reducing a problem to evaluations of upper envelopes is known as the *convex hull trick* [2].

For the sake of readability, we mirror x and all indices so that from now on we consider $f_1, \dots, f_n$ and $x_1, \dots, x_n$ with $f_i(x) := F_{n+1-i}(-x)$ and $x_i := -p_x^{n-i}$. Then we have $x_1 < \dots < x_n$ and $\text{slope}(f_1) < \dots < \text{slope}(f_n)$, and the goal is to compute $h_i = \max_{1\leqslant j\leqslant i} f_j(x_i)$, which equals the desired value H_i^ℓ. Since each function f_j describes a line, the function

$$U_i(x) := \max_{1\leqslant j\leqslant i} f_j(x)$$

describes the *upper envelope* of those lines. Hence, our goal is to evaluate (incremental) upper envelopes of lines at given positions. Since U_i is an upper envelope, it describes an x-monotone polygonal chain (since the part of U_i, where a particular f_j is maximal, is a line segment). By following the upper envelope from left to right, we first follow $f_{j(1)}$, then $f_{j(2)}$, and so on, until we finally follow $f_{j(h)}$. We represent this upper envelope by the sequence $(f_{j(1)}, \dots, f_{j(h)})$. With slight abuse of notation we write $U_i = (f_{j(1)}, \dots, f_{j(h)})$.

Computing the upper envelope of lines is a standard task in computational geometry and can be done in linear time (since our functions f_j are already sorted by slope) [11]. You can find the standard algorithm for this task by considering lines 1–5 of Algorithm 2. Here, we use a standard dequeue S containing linear functions, and we denote the elements of S by $S_1, \dots, S_{|S|}$. Moreover, we denote the x-coordinate of the intersection of two linear functions f and g by $\otimes(f, g)$; this can be computed in constant time. If we deleted lines 6–8 from Algorithm 2, then after the i-th iteration the dequeue S would contain exactly the upper envelope U_i, in our representation as a sequence $(f_{j(1)}, \dots, f_{j(h)})$.

Through lines 6–8 of Algorithm 2 we have adapted the standard upper envelope algorithm as follows. In every iteration, we search for the element of the upper envelope $U_i = (f_{j(1)}, \dots, f_{j(h)})$ that is maximal at position x_i. Say, $f_{j(v)}(x_i)$ is maximal among all $f_j(x_i)$, $1 \leqslant j \leqslant i$. Since $x_1, \dots, x_n$ are sorted ascendingly, any function $f_{j(v')}$, $v' < v$, can be deleted, since it will not be of importance to any x_j, $j > i$. Finally, we evaluate $f_{j(v)}$ at x_i, which yields h_i. This explains lines 6–8 of Algorithm 2.

Runtime. The runtime of Algorithm 2 is linear, since we can amortize the append() operations and deletions. There are exactly n append() operations, since in each iteration of the loop we add one element to S. Thus, we can perform at most n deletions in total, and the overall runtime is $\mathcal{O}(n)$.

Correctness. Intuitively, we prove that in the i-th iteration the dequeue S contains all functions (in sorted order) that appear in the upper envelope of $f_1, \dots, f_i$ restricted to the halfplane $\{(x, y) \in \mathbb{R}^2 \mid x \geqslant x_i\}$. Thus, indeed the first entry S_1 corresponds to the segment of the upper envelope of $f_1, \dots, f_i$ that contains x_i, so we correctly compute h_i.

In the previous section we defined $U_i = (f_{j(1)}, \dots, f_{j(h)})$ as the upper envelope of $f_1, \dots, f_i$. Note that the function $f_{j(v)}$ is maximal among all f_j for any x between the intersections of $f_{j(v-1)}$ and $f_{j(v)}$ and the intersection of $f_{j(v)}$ and $f_{j(v+1)}$. More formally, for all $1 < v < h$ the value $f_{j(v)}(x)$ is maximal among all $f_j(x)$ for any $x \in [\otimes(f_{j(v-1)}, f_{j(v)}), \otimes(f_{j(v)}, f_{j(v+1)})] =: I_v(U_i)$, and $f_{j(1)}(x)$ is maximal for any $x \in (-\infty, \otimes(f_{j(1)}, f_{j(2)})] =: I_1(U_i)$, and $f_{j(h)}(x)$ is maximal for any $x \in [\otimes(f_{j(h-1)}, f_{j(h)}), \infty) =: I_h(U_i)$. This way, the real line is partitioned into $I_1(U_i) \cup \dots \cup I_h(U_i)$, such that function $f_{j(v)}$ is maximal among all f_j, $1 \leqslant j \leqslant i$, in interval $I_v(U_i)$.

For proving correctness we show that at the end of the i-th iteration the following invariant holds.

Lemma 5. *Let $U_i = (f_{j(1)}, f_{j(2)}, \dots, f_{j(h)})$ be the upper envelope of $f_1, \dots, f_i$ and let a be such that $x_i \in I_a(U_i)$. Then after the end of the i-th iteration the dequeue S is*

$$S = (f_{j(a)}, f_{j(a+1)}, \dots, f_{j(h)}).$$

Note that this invariant implies correctness of the computed value h_i: Since $x_i \in I_a(U_i)$, function $f_{j(a)}$ is maximal among $f_1, \dots, f_i$ at position x_i. Hence, we have

$$h_i = \max_{1\leqslant j\leqslant i} f_j(x_i) = f_{j(a)}(x_i) = S_1(x_i).$$

Proof. We argue that the invariant holds true. By induction, we can assume that it is true at the end of iteration $i - 1$, so that $S = (f_{j(a)}, f_{j(a+1)}, \dots, f_{j(h)})$, where $U_{i-1} = (f_{j(1)}, f_{j(2)}, \dots, f_{j(h)})$ is the upper envelope of $f_1, \dots, f_{i-1}$, and $x_{i-1} \in I_a(U_{i-1})$. We show that it also holds after iteration i.

Observe that $S = (f_{j(a)}, \dots, f_{j(h)})$, being a subsequence of an upper envelope, is itself the upper envelope of $\{f_{j(a)}, f_{j(a)+1} \dots, f_{i-2}, f_{i-1}\}$. Hence, lines 3–5 of Algorithm 2 correctly compute the upper envelope $U' = (f_{j'(1)}, f_{j'(2)}, \dots, f_{j'(g)})$ of $\{f_{j(a)}, f_{j(a)+1} \dots, f_{i-1}, f_i\}$, since this is the well-known algorithm for computing upper envelopes (for a proof of correctness of this algorithm we essentially show that we can delete $S_{|S|}$ if and only if it lies underneath f_i in the interval $(\otimes(S_{|S|}, S_{|S|-1}), \infty)$, and this happens if and only if $\otimes(f_i, S_{|S|-1}) \leqslant \otimes(S_{|S|}, S_{|S|-1})$).

Since we know that $f_1, \dots, f_{j(a)-1}$ are less than $f_{j(a)}$ at position $x_{i-1} \leqslant x_i$, they are also less than $f_{j(a)}$ at position x_i, since the slopes are increasing. Hence, the maximum of $f_1, \dots, f_i$ at x_i is attained by $U' = (f_{j'(1)}, f_{j'(2)}, \dots, f_{j'(g)})$. It remains to find the interval of U' containing x_i, which can be done by throwing away all $f_{j'(v)}$ with $\otimes(f_{j'(v)}, f_{j'(v+1)}) \leqslant x_i$, since $f_{j'(v+1)}$ is bigger than $f_{j'(v)}$ to the right of $\otimes(f_{j'(v)}, f_{j'(v+1)})$, so that it is also bigger at x_i. This is done by lines 6–7 of Algorithm 2. More formally, let b such that $x_i \in I_b(U')$. Then in lines 6–7 we delete the functions $f_{j'(1)}, \dots, f_{j'(b-1)}$ from $S = U' = (f_{j'(1)}, f_{j'(2)}, \dots, f_{j'(g)})$, thereby restoring the invariant. □

3.3 Reconstruction of P^*

As is the case for many dynamic-programming-like algorithms, from the computation of the optimal value $\mathcal{I}_{\text{hyp}}(P^*)$ we can efficiently reconstruct an optimal solution P^* achieving the optimal value $\mathcal{I}_{\text{hyp}}(P^*)$.

To this end, we augment our dequeue S to not only store linear functions f_j, but tuples (f_j, j), so that we know the index j of every linear function in S. With this, in the computation of H_i^ℓ we not only compute its value, but also find some $j > i$ such that $H_i^\ell = F_j^\ell(p_x^i)$. This tells us that when we selected point p^i and want to select ℓ more points, then the next point should be j. Let us store

$$\text{Next}_i^\ell := j.$$

Then an optimal solution P^* can be reconstructed as

$$\{p^{i_1}, \dots, p^{i_k}\}$$

by setting $i_0 := 0$ and $i_{\ell+1} := \text{Next}_{i_\ell}^{k-\ell}$.

We remark that this increases the space requirement from $\mathcal{O}(n)$ to $\mathcal{O}(k\,n)$, as we now have to store the full table Next.

4. ALGORITHM FOR EPSSSP

We start with an outline of the algorithm. Let (k, P, R) be a EPSSSP instance with (unknown) optimal value ε^*. We assume that P and R are non-dominating and sorted. We first design a test to check for any given $\varepsilon \in \mathbb{R}$ whether $\varepsilon \geqslant \varepsilon^*$. In other words, we want to check whether there exists a subset $P' \subseteq P$ of size k with $\mathcal{I}_{\text{eps}}(P', R) \leqslant \varepsilon$. We call this an *$\varepsilon$-test*. Extending an algorithm by Koltun and Papadimitriou [15] (who consider the case $P = R$) we show the following lemma.

Lemma 6. *An ε-test can be performed in time $\mathcal{O}(n + m)$.*

We could try to use ε-tests directly for a binary search for the optimal approximation ratio $\varepsilon^* \in \mathbb{R}$. This could yield arbitrarily good approximations to ε^*, but would not allow to compute ε^* exactly, as it can be any real number. However, ε^* is by definition among the $n \cdot m$ values $S = \{\mathcal{I}_{\text{eps}}(p, r) \mid p \in P,\ r \in R\}$. We obtain a simple $\mathcal{O}(nm \log(nm))$ time algorithm by sorting S and then doing a binary search for ε^* in S, incurring $\log(nm)$ ε-tests, see Algorithm 3.

Algorithm 3: EPSSSP in 2 dimensions, $\mathcal{O}(nm \log(nm))$ time

```
Input: integer $k$, non-dominating and sorted sets $P$, $R$
1 sort $S = \{\mathcal{I}_{\text{eps}}(p, r) \mid p \in P,\ r \in R\}$
2 binary search for $\varepsilon^* \in S$ using $\varepsilon$-tests (see Algorithm 4)
```

To get quasi-linear runtime, we observe that one can partition the possible values for ε^* into $2n$ sorted sequences $S_1, \dots, S_{2n}$ of length at most m each. Note that running a binary search on each sorted sequence using ε-tests still yields a runtime of $\Omega(nm)$, which is not quasilinear. Instead, we search over the sorted sequences S_i with a more intelligent randomized algorithm as follows. In each iteration we take a random pivot element ε_p from $S = \bigcup_i S_i$. We do an ε-test with ε_p. Say, we get the answer $\varepsilon_p \geqslant \varepsilon^*$. This allows to delete all values in the sequences S_i that are at least ε_p. Deleting these values takes time $\mathcal{O}(n \log m)$: in every sequence S_i we do a binary search for ε_p to determine the cutoff point. Similarly, if the answer of the ε-test was $\varepsilon_p < \varepsilon^*$ then we can delete all values in the sequences S_i that are at most ε_p in time $\mathcal{O}(n \log m)$. The crucial observation now is that the expected size of $\bigcup_i S_i$ drops by a constant factor in each iteration. Since in the beginning we have $|\bigcup_i S_i| = \mathcal{O}(nm)$, after $\mathcal{O}(\log(nm))$ iterations (in expectation and with high probability[1]) we have deleted all elements of the sequences S_i and found ε^* on the way. This yields a total runtime of $\mathcal{O}((n \log m + m) \log(nm))$.

We present an improvement to runtime $\mathcal{O}((n + m) \log(nm)) = \mathcal{O}(n \log n + m \log m)$ in Section 4.3, which is more of theoretical interest, as indicated by our preliminary experiments.

In the remainder of this section, we work out these ideas in detail.

4.1 ε-Test

In this section we prove Lemma 6. Recall that, given an EPSSSP instance (k, P, R) with (unknown) optimal value ε^* and given $\varepsilon \in \mathbb{R}$, an ε-test determines whether $\varepsilon \geqslant \varepsilon^*$. We present a greedy algorithm with runtime $\mathcal{O}(n + m)$ (cf. Algorithm 4).

Recall that $P = \{p^1, \dots, p^n\}$ and $R = \{r^1, \dots, r^m\}$ are already sorted and non-dominating, so that they are sorted by increasing x-coordinates as well as decreasing y-coordinates.

The rightmost point r^m of R (i.e., the one with largest x-coordinate) has to be approximated by some point in the chosen subset $P' \subseteq P$. Consider the leftmost point p^i in P with $p_x^i + \varepsilon \geqslant r_x^m$. The point p^i approximates r^m in x-direction. If it does not approximate r^m in y-direction, then *no* point in P ε-approximates r^m, since, among all points $p \in P$ with sufficiently large x-coordinate to approximate r^m, p^i is the point with largest y-coordinate (because it is leftmost). Thus, in this case we can safely answer $\varepsilon < \varepsilon^*$.

If, on the other hand, p^i ε-approximates r^m then we may greedily pick p^i into our selected set $P' \subseteq P$. To argue this, we show that p^i ε-approximates a superset of any other point p^h that ε-approximates r^m. Formally, for any h with $\mathcal{I}_{\text{eps}}(p^h, r^m) \leqslant \varepsilon$ we have $\{r \in R \mid \mathcal{I}_{\text{eps}}(p^h, r) \leqslant \varepsilon\} \subseteq \{r \in R \mid \mathcal{I}_{\text{eps}}(p^i, r) \leqslant \varepsilon\}$. Note that $h \geqslant i$, since p^i is the leftmost point ε-approximating r^m. Now, if $r \in R$ is ε-approximated by p^h then $p_y^i + \varepsilon \geqslant p_y^h + \varepsilon \geqslant r_y$ (since $h \geqslant i$ and p^h ε-approximates r) and $p_x^i + \varepsilon \geqslant r_x^m \geqslant r_x$ (since p^i ε-approximates r^m and r^m is rightmost in R). Thus, p^i also ε-approximates r. Hence, if there is at all a set $P' \subseteq P$ of size k that ε-approximates R, then there is such a set containing p^i, and we may greedily pick p^i to be included in P' without violating optimality.

Now consider the leftmost point r^j with $r_y^j \leqslant p_y^i + \varepsilon$. Since we also have $p_x^i + \varepsilon \geqslant r_x^m \geqslant r_x^j$, the points $r^j, r^{j+1}, \dots, r^m$ are ε-approximated by p^i, and these are all points that are ε-approximated by p^i. Since these points are already covered by picking p^i, we may delete them from R, i.e., set $m := j - 1$. Moreover, since p^i approximates a superset of all points to its right, we may delete $p^i, p^{i+1}, \dots, p^n$ from P, i.e., set $n := i - 1$; these points no longer approximate any points in R. We are left with a new instance of EPSSSP on new sets P, R with k reduced by one, so we may repeat this procedure.

[1]I.e., with probability $\geqslant 1 - n^{-c}$ for any $c > 0$, where the constant hidden in the $\mathcal{O}$-notation of the number of iterations depends on c.

For termination, we know that $\varepsilon \geqslant \varepsilon^*$ if after choosing any $\ell \leqslant k$ points we have $m = 0$, meaning that the points picked so far ε-approximate all points in R already. Moreover, if after choosing k points we do not yet approximate all of R (i.e., $m > 0$), then we have $\varepsilon < \varepsilon^*$.

Algorithm 4: ε-Test

```
Input: real $\varepsilon$, integer $k$, non-dominating and sorted sets
       $P = \{p^1, \dots, p^n\}$, $R = \{r^1, \dots, r^m\}$
1  $P' := \emptyset$
2  for $\ell \leftarrow 1$ to $k$ do
3      let $i$ be the minimal $i \in \{1, \dots, n\}$ with $p_x^i + \varepsilon \geqslant r_x^m$
4      if $\mathcal{I}_{\text{eps}}(p^i, r^m) > \varepsilon$ then return "$\varepsilon < \varepsilon^*$"
5      $P' := P' \cup \{p^i\}$
6      let $j$ be the minimal $j \in \{1, \dots, m\}$ with $r_y^j \leqslant p_y^i + \varepsilon$
7      $m := j - 1$    // delete $r^j, \dots, r^m$ from $R$
8      $n := i - 1$    // delete $p^i, \dots, p^n$ from $P$
9      if $m = 0$ then return "$\varepsilon \geqslant \varepsilon^*$"
10 return "$\varepsilon < \varepsilon^*$"
```

We now argue that Algorithm 4 can be implemented in time $\mathcal{O}(n + m)$. To this end, we implement the search for i in line 3 by a linear scan over $n, n-1, \dots, i+1, i$. Observe that every element that is touched by this linear scan is deleted later in the same iteration. Thus, in total we touch every point in P at most once. Similarly, we search for j by considering $m, m-1, \dots, j+1, j$, so that we touch every point in R at most once. This yields a runtime bound of $\mathcal{O}(n + m)$.

4.2 Subset Selection

In this section, we show how to use the ε-test from the last section to solve EPSSSP in time $\mathcal{O}((n \log m + m) \log(nm))$, proving a weaker version of Theorem 2.

Sorted Sequences. As has been observed in [18, 19], ε^* is among the values $\mathcal{I}_{\text{eps}}(p, r)$ with $p \in P$, $r \in R$, since at least one pair (p, r) prevents us from further decreasing ε^*. We can split these values into sorted sequences as follows. Write $R = \{r^1, \dots, r^m\}$ and recall that this sequence of points is sorted by increasing x-coordinate and decreasing y-coordinate. Observe that this implies that R is also sorted according to increasing $r_x - r_y$ values. Now fix a point $p \in P$. Determine the largest index $1 \leqslant i_p \leqslant m$ such that $r_x^{i_p} - r_y^{i_p} \leqslant p_x - p_y$. Since R is sorted by $r_x - r_y$, a binary search does this in time $\mathcal{O}(\log m)$. Then the following sequences are sorted (this is essentially the same property as [19, Proposition 3.2]).

Lemma 7. *For any $p \in P$, the following sequences are sorted:*

$$S_p^y := \left(\mathcal{I}_{\text{eps}}(p, r^1), \dots, \mathcal{I}_{\text{eps}}(p, r^{i_p})\right),$$
$$S_p^x := \left(\mathcal{I}_{\text{eps}}(p, r^{i_p+1}), \dots, \mathcal{I}_{\text{eps}}(p, r^m)\right).$$

Proof. Since R is sorted by increasing values of $r_x - r_y$ and i_p is the largest index with $r_x^{i_p} - r_y^{i_p} \leqslant p_x - p_y$, we have $r_x^i - r_y^i \leqslant p_x - p_y$ for all $1 \leqslant i \leqslant i_p$ and $r_x^i - r_y^i > p_x - p_y$ for all $i_p + 1 \leqslant i \leqslant m$. Let us focus on $1 \leqslant i \leqslant i_p$, or equivalently the sequence S_p^y. Note that $r_x^i - r_y^i \leqslant p_x - p_y$ is equivalent to $r_x^i - p_x \leqslant r_y^i - p_y$. Hence, we have $\mathcal{I}_{\text{eps}}(p, r^i) = \max\{r_x^i - p_x, r_y^i - p_y\} = r_y^i - p_y$. Since p is fixed and $\{r^1, \dots, r^{i_p}\}$ is sorted by y-coordinates, also the values $r_y^i - p_y$ are sorted, proving that the sequence S_p^y is sorted. An analogous argument shows the claim for S_p^x. □

Note that the sequences S_p^x, S_p^y, $p \in P$, form $2n$ sequences with at most m elements each. We will denote these sequences by $S_1, \dots, S_{2n}$ from now on. These sequences are all of the form $\{\mathcal{I}_{\text{eps}}(p, r^s), \mathcal{I}_{\text{eps}}(p, r^{s+1}), \dots, \mathcal{I}_{\text{eps}}(p, r^t)\}$ for some $p \in P$ and $1 \leqslant s, t \leqslant m$. Note that we can store such a sequence implicitly by only storing p, s, t, and do not have to explicitly store all its elements. This implicit representation allows us to efficiently support the following operations on $S_i = (p_i, s_i, t_i)$:

(1) Compute the j-th element: This is $\mathcal{I}_{\text{eps}}(p_i, r^{s_i+j-1})$.
(2) Compute the size $|S_i|$: This is $t_i - s_i + 1$.
(3) Delete the first j elements: Change S_i to $(p_i, s_i + j, t_i)$.
(4) Delete the last j elements: Change S_i to $(p_i, s_i, t_i - j)$.

As a final remark, note that some of the sequences $S_1, \dots, S_{2n}$ are ordered ascendingly and some descendingly. For the sake of readability, we want to assume from now on that *all sequences $S_1, \dots, S_{2n}$ are sorted in increasing order*. To achieve this, we may augment the implicit representation of a sequence by a bit specifying whether it is sorted ascendingly or descendingly. In case it is sorted descendingly, we then mirror indices appropriately, e.g., when accessing the j-th element of $S_i = (p_i, s_i, t_i)$ that is sorted descendingly we return $\mathcal{I}_{\text{eps}}(p_i, r^{t_i-j+1})$.

In summary, we obtain $2n$ sequences $S_1, \dots, S_{2n}$ sorted ascendingly and of size at most m each, such that ε^* is among these sequences. Computing (an implicit representation of) these sequences takes time $\mathcal{O}(n \log m)$, see also Algorithm 5.

Algorithm 5: Computing sorted sequences

```
Input: non-dominating, sorted sets $P$ and $R = \{r^1, \dots, r^m\}$
1  foreach $p \in P$ do
2      binary search for the largest $1 \leqslant i_p \leqslant m$ with
3          $r_x^{i_p} - r_y^{i_p} \leqslant p_x - p_y$
4  the sorted sequences are $\{\mathcal{I}_{\text{eps}}(p, r^1), \dots, \mathcal{I}_{\text{eps}}(p, r^{i_p})\}$
   (represented as $(p, 1, i_p)$) and
   $\{\mathcal{I}_{\text{eps}}(p, r^{i_p+1}), \dots, \mathcal{I}_{\text{eps}}(p, r^m)\}$ (represented as
   $(p, i_p + 1, m)$) for each $p \in P$.
```

Binary Search over Many Sequences. We are left with the following abstract problem. Given sorted sequences $S_1, \dots, S_{2n}$ of length at most m containing an unknown target value ε^*, and access to an ε-test that computes for a given ε whether $\varepsilon \geqslant \varepsilon^*$ and runs in time $\mathcal{O}(n + m)$, compute ε^*.

Denote by $S := \bigcup_i S_i$ the concatenation of the sequences S_i. In every iteration we pick a uniformly random pivot element ε_p in S (which can be done in $\mathcal{O}(n)$ time) and do an ε-test with ε_p. If $\varepsilon_p \geqslant \varepsilon^*$ we memorize ε_p as a candidate for ε^*; we only have to store the smallest among all candidates. Then we delete all values $\varepsilon \geqslant \varepsilon_p$ in all the sequences S_i (as we already found a better candidate for ε^* than all these values). Note that these deletions can be done in $\mathcal{O}(\log m)$ time per sequence S_i by running a binary search for ε_p in S_i and then deleting all greater elements – all necessary operations are supported by our implicit representation of the sequences S_i. Otherwise, if $\varepsilon_p < \varepsilon^*$ then we instead delete all values $\varepsilon \leqslant \varepsilon_p$ (as all these values are smaller than ε^*). The crucial property now is the following:

> In each iteration, with constant probability we reduce the size of S by a constant factor. (*)

Since $|S| = \mathcal{O}(nm)$ in the beginning, this property implies that after $\mathcal{O}(\log(nm))$ iterations (in expectation and with high probability) the set S is empty. Since we must have deleted ε^* at some point, our current candidate is equal to ε^*, and the algorithm has a correct output, see Algorithm 6.

Since in each iteration we do one ε-test and $\mathcal{O}(n)$ binary searches in the S_i's, we obtain a total runtime of $\mathcal{O}((n \log m + m)\log(nm))$ (in expectation and with high probability).

Algorithm 6: EPSSSP in 2 dimensions, slower version

```
Input: integer $k$, non-dominating and sorted sets $P$, $R$
1  compute sorted sequences $S_1, \dots, S_{2n}$ containing $\varepsilon^*$ using
   Algorithm 5
2  $\varepsilon_{\text{cand}} := \infty$
3  while not all sets $S_i$ empty do
4      choose $\varepsilon_p$ uniformly at random over all sequences $S_i$
5      $\varepsilon$-test with $\varepsilon_p$ using Algorithm 4
6      if $\varepsilon_p \geqslant \varepsilon^*$ then
7          $\varepsilon_{\text{cand}} := \min\{\varepsilon_{\text{cand}}, \varepsilon_p\}$
8          for $1 \leqslant i \leqslant n$ do
9              delete all $\varepsilon \geqslant \varepsilon_p$ from $S_i$ using a binary search
10     else
11         for $1 \leqslant i \leqslant n$ do
12             delete all $\varepsilon \leqslant \varepsilon_p$ from $S_i$ using a binary search
13 return $\varepsilon_{\text{cand}}$
14 the corresponding set of points $P^*$ is the set computed by an
   $\varepsilon$-test with $\varepsilon = \varepsilon_{\text{cand}}$
```

Proof of Property (*) for Algorithm 6. Denote by U_i the number of elements of S_i that are at least ε_p, and by L_i the number of elements of S_i that are smaller than ε_p. Let

$$U := \sum_i U_i \quad \text{and} \quad L := \sum_i L_i.$$

Observe that, independent of the value of ε^*, we delete at least $\min\{U, L+1\}$ elements from $S = \bigcup_i S_i$: If $\varepsilon_p \geqslant \varepsilon^*$ we delete the U elements that are at least ε_p, and if $\varepsilon_p < \varepsilon^*$ we delete the L smaller elements and ε_p itself.

Since we pick ε_p uniformly at random in S, we pick ε_p in the middle third of S with probability 1/3. More precisely, we have $\lceil \frac{1}{3}|S| \rceil \leqslant U \leqslant \lceil \frac{2}{3}|S| \rceil$ with probability $\frac{1}{|S|}(\lceil \frac{2}{3}|S| \rceil - \lceil \frac{1}{3}|S| \rceil + 1) \geqslant \frac{1}{3}$. If this event occurs, then $\min\{U, L+1\} \geqslant \frac{1}{3}|S|$. Thus, with constant probability we delete at least a constant fraction of S. □

4.3 Theoretical Improvement

In this section, we reduce the runtime to $\mathcal{O}(n \log n + m \log m)$. To this end, we show that for having Property (*) it is not necessary to do a full binary search for the pivot ε_p in each sequence S_i.

Again, in each iteration of the algorithm we choose a uniformly random pivot ε_p in $S = \bigcup_i S_i$ and do an ε-test with ε_p. Assume first that $\varepsilon_p \geqslant \varepsilon^*$. Again we store ε_p as a candidate in this case. Now we consider the $\lceil \frac{2}{3}|S_i| \rceil$-th element ε_i of S_i; as S_i is sorted we can access this element in constant time. If $\varepsilon_i \geqslant \varepsilon_p$ then we delete all elements in S_i to the right of ε_i (including ε_i, these values are at least as large as our current candidate). Otherwise S_i stays as is. If, on the other hand, $\varepsilon_p < \varepsilon^*$ then we consider the $\lceil \frac{1}{3}|S_i| \rceil$-th element ε_i of S_i. If $\varepsilon_i \leqslant \varepsilon_p$ then we delete all elements to the left of ε_i (including ε_i). Otherwise S_i stays as is. Note that now an iteration only takes time $\mathcal{O}(n+m)$. Again, we repeat these iterations until all sequences are empty, in this case the candidate we stored for ε^* is finally correct. These changes result in Algorithm 7.

Algorithm 7: EPSSSP in 2 dimensions, faster version

```
Input: integer $k$, non-dominating and sorted sets $P$, $R$
The algorithm is the same as Algorithm 6, except that we
replace line 9 by:
9a         let $\varepsilon_i$ be the $\lceil \frac{2}{3}|S_i| \rceil$-th element of $S_i$
9b         if $\varepsilon_i \geqslant \varepsilon_p$ then
9c             delete $\varepsilon_i$ and all elements to its right from $S_i$
replace line 12 by:
12a        let $\varepsilon_i$ be the $\lceil \frac{1}{3}|S_i| \rceil$-th element of $S_i$
12b        if $\varepsilon_i \leqslant \varepsilon_p$ then
12c            delete $\varepsilon_i$ and all elements to its left from $S_i$
```

To bound the runtime of Algorithm 7, it remains to show that Property (*) is still satisfied. If this is the case, then we again need $\mathcal{O}(\log(nm))$ iterations and obtain a total runtime of $\mathcal{O}((n+m)\log(nm)) = \mathcal{O}(n \log n + m \log m)$ (in expectation and with high probability).

Proof of Property (*) for Algorithm 7. Again, let $U_i := \{s \in S_i \mid s \geqslant \varepsilon_p\}$, $L_i := \{s \in S_i \mid s < \varepsilon_p\}$ and $U := \sum_i U_i$, $L := \sum_i L_i$.

Claim 8. *In the current iteration, Algorithm 7 deletes at least* $\min\{\frac{1}{3}U - \frac{1}{6}L, \frac{1}{3}(L+1) - \frac{1}{6}(U-1)\}$ *elements.*

Proof. First assume that $\varepsilon_p \geqslant \varepsilon^*$. We prove that in any set S_i we delete at least $\frac{1}{3}U_i - \frac{1}{6}L_i$ elements. Summing over all $1 \leqslant i \leqslant 2n$ then yields the result. Consider Algorithm 7 and note that if $\varepsilon_i \geqslant \varepsilon_p$ then we delete all elements to the right of ε_i (including ε_i). Thus, we delete at least $\frac{1}{3}|S_i|$ elements. As $U_i \leqslant |S_i|$ and $L_i \geqslant 0$, we delete at least

$$\tfrac{1}{3}|S_i| \geqslant \tfrac{1}{3}U_i - \tfrac{1}{6}L_i$$

elements. If, on the other hand, $\varepsilon_i < \varepsilon_p$, then we do not delete any elements from S_i. In this case we have $L_i \geqslant \frac{2}{3}|S_i|$ and $U_i \leqslant \frac{1}{3}|S_i|$, so that

$$0 \geqslant \tfrac{1}{3}U_i - \tfrac{1}{6}L_i.$$

Hence, also in this case we delete at least $\frac{1}{3}U_i - \frac{1}{6}L_i$ elements, because this number is non-positive.

In summary, we obtain that if $\varepsilon_p \geqslant \varepsilon^*$ we delete at least $\frac{1}{3}U - \frac{1}{6}L$ elements. In the case $\varepsilon_p < \varepsilon^*$, a symmetric argument using $L' := \{s \in S \mid s \leqslant \varepsilon_p\}$ and $U' := \{s \in S \mid s > \varepsilon_p\}$ shows that we delete at least $\frac{1}{3}L' - \frac{1}{6}U'$ elements of S. Recalling the definition of $L = \{s \in S \mid s < \varepsilon_p\}$ and $U = \{s \in S \mid s \geqslant \varepsilon_p\}$, we see that $L' \geqslant L+1$ and $U' \leqslant U-1$. Hence, in this case we delete at least $\frac{1}{3}(L+1) - \frac{1}{6}(U-1)$ elements of S. □

To finish the proof of Property (*), note that since ε_p is uniformly random in S we have $\lceil \frac{2}{5}|S| \rceil \leqslant U \leqslant \lceil \frac{3}{5}|S| \rceil$ with probability at least $\frac{1}{5}$. If this event occurs, then $\min\{\frac{1}{3}U - \frac{1}{6}L, \frac{1}{3}(L+1) - \frac{1}{6}(U-1)\} \geqslant \frac{1}{3} \cdot \frac{2}{5}|S| - \frac{1}{6} \cdot \frac{3}{5}|S| = \frac{1}{30}|S|$. Thus, with constant probability we delete at least a constant fraction of S. □

5. EXPERIMENTS

All the presented algorithms for the hypervolume indicator as well as the epsilon indicator have been implemented and are available at the homepage of the second author [1].

We experimentally compared our algorithm for HYPSSP with runtime $\mathcal{O}(n \cdot k)$ (Algorithm 1) with the dynamic programming approach of Auger et al. [3] with runtime $\mathcal{O}(n^2 k)$. For the ε-indicator we set $P = R$ so that $n = m$. We implemented the $\mathcal{O}(n^2 \log n)$ algorithm (Algorithm 3), which has the same asymptotic runtime as [18, 19], and compared it with both algorithms we described (Algorithm 6 and Algorithm 7).

We tested instances for increasing $10^2 \leqslant n \leqslant 10^7$ and $k \in \{10, 20, 50, n/2\}$. These values for k have been chosen with the motivations in mind that we sketched in the introduction. The algorithm runtimes was averaged over 30 runs for n randomly sampled points or points with equal distance taken from the fronts $f_1(x) = 1 - x$, $f_2(x) = \sqrt{1 - x^2}$, and $f_3(x) = \frac{1}{x}$.

All experiments showed a similar behavior. We observed no differences in the algorithms' runtime behavior for the three fronts and the two ways of sampling. The measured runtimes fit very well to the asymptotic guarantees, meaning that all hidden constants are reasonably small. However, Algorithm 6 was as fast as Algorithm 7 in our experiments, i.e., this minor runtime improvement is only of theoretical nature. The experiments showed that our new algorithms are empirically faster by several orders of magnitude than the previous state of the art. Details will be provided in an extended version of this paper.

Acknowledgments

The research leading to these results has received funding from the Australian Research Council (ARC) under grant agreement DP140103400 and from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 618091 (SAGE). Karl Bringmann is a recipient of the *Google Europe Fellowship in Randomized Algorithms*, and this research is supported in part by this Google Fellowship.

References

[1] `http://docs.theinf.uni-jena.de/code/ssp.zip`.

[2] `http://wcipeg.com/wiki/Convex_hull_trick`.

[3] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler. Investigating and exploiting the bias of the weighted hypervolume to articulate user preferences. In *11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*, pp. 563–570. ACM Press, 2009.

[4] J. M. Bader. *Hypervolume-Based Search for Multiobjective Optimization: Theory and Methods*. PhD thesis, Eidgenössische Technische Hochschule ETH Zürich, 2009.

[5] N. Beume, B. Naujoks, and M. T. M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181:1653–1669, 2007.

[6] N. Beume, C. M. Fonseca, M. López-Ibáñez, L. Paquete, and J. Vahrenhold. On the complexity of computing the hypervolume indicator. *IEEE Trans. Evolutionary Computation*, 13:1075–1082, 2009.

[7] N. Beume, B. Naujoks, M. Preuss, G. Rudolph, and T. Wagner. Effects of 1-greedy $\mathcal{S}$-metric-selection on innumerably large Pareto fronts. In *5th International Conference on Evolutionary Multi-Criterion Optimization (EMO '09)*, Vol. 5467 of *LNCS*, pp. 21–35, 2009.

[8] K. Bringmann and T. Friedrich. An efficient algorithm for computing hypervolume contributions. *Evolutionary Computation*, 18:383–402, 2010.

[9] K. Bringmann, T. Friedrich, F. Neumann, and M. Wagner. Approximation-guided evolutionary multi-objective optimization. In *22nd International Joint Conference on Artificial Intelligence (IJCAI '11)*, pp. 1198–1203. IJCAI/AAAI, 2011.

[10] P. Brucker. Efficient algorithms for some path partitioning problems. *Discrete Applied Mathematics*, 62:77–85, 1995.

[11] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry*. Springer, 2008.

[12] M. T. M. Emmerich and C. M. Fonseca. Computing hypervolume contributions in low dimensions: Asymptotically optimal algorithm and complexity results. In *6th International Conference on Evolutionary Multi-Criterion Optimization (EMO '11)*, Vol. 6576 of *LNCS*, pp. 121–135. Springer, 2011.

[13] C. Igel, N. Hansen, and S. Roth. Covariance matrix adaptation for multi-objective optimization. *Evolutionary Computation*, 15:1–28, 2007.

[14] C. Igel, T. Suttorp, and N. Hansen. Steady-state selection and efficient covariance matrix update in the multi-objective CMA-ES. In *4th International Conference on Evolutionary Multi-Criterion Optimization (EMO '07)*, Vol. 4403 of *LNCS*, pp. 171–185. Springer, 2007.

[15] V. Koltun and C. H. Papadimitriou. Approximately dominating representatives. *Theoretical Computer Science*, 371:148–154, 2007.

[16] T. Kuhn, C. M. Fonseca, L. Paquete, S. Ruzika, and J. R. Figueira. Hypervolume subset selection in two dimensions: Formulations and algorithms. Technical report, Fachbereich Mathematik, TU Kaiserslautern, 2014.

[17] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22: 469–476, 1975.

[18] A. Ponte, L. Paquete, and J. R. Figueira. On beam search for multicriteria combinatorial optimization problems. In *9th International Conference in Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems (CPAIOR '12)*, Vol. 7298 of *LNCS*, pp. 307–321. Springer, 2012.

[19] D. Vaz, L. Paquete, and A. Ponte. A note on the ε-indicator subset selection. *Theoretical Computer Science*, 499: 113–116, 2013.

[20] E. Zitzler, D. Brockhoff, and L. Thiele. The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration. In *4th International Conference on Evolutionary Multi-Criterion Optimization (EMO '07)*, Vol. 4403 of *LNCS*, pp. 862–876. Springer, 2007.