Data Structures for Incremental Extreme Ray Enumeration Algorithms

Blagoy Genov*

Abstract

Given a halfspace $\mathcal{H}$ and a polyhedral cone $\mathcal{P}$ with a known extreme ray set $\mathcal{V}$ we consider the problem of finding the extreme ray set for the cone $\mathcal{P}' = \mathcal{H} \cap \mathcal{P}$. Regarding the computational time of the above problem, best results have been achieved with data structures based on multidimensional binary search trees. We refined the existing algorithm by developing a specific method for tree creation which brought further computational speedup. Furthermore, we examined alternative data structures based on vantage point trees and identified potential scenarios for their application.

1 Introduction

Polyhedral Cones. A nonempty set $\mathcal{P}$ of vectors in $\mathbb{R}^d$ is called a *(convex) polyhedral cone* if there exists a nonzero *representation matrix* $A \in \mathbb{R}^{n \times d}$ such that

$$\mathcal{P} := \{x \in \mathbb{R}^d : Ax \geq 0\}.$$

Those vectors $\mathcal{V}^E \subseteq \mathcal{P}$ which cannot be expressed as a conical combination of other vectors are called *extreme rays* of $\mathcal{P}$. The *active set* of extreme rays is defined by means of a mapping

$$\psi : \mathcal{V}^E \to \{0,1\}^n$$

from extreme rays to binary vectors: $z = \psi(r)$ identifies the row vectors of A which r satisfied with equality, in the sense that

$$z_i = 0 \Rightarrow a_i^T \cdot r > 0 \text{ and } z_i = 1 \Rightarrow a_i^T \cdot r = 0$$

where a_i is the i-th row vector of A. Given the binary vectors $z = (z_1, \ldots, z_n)$ and $z' = (z'_1, \ldots, z'_n)$ the operations $\wedge$ and $\bar{z}$ (complement) as well as the relations $\leq$ and $<$ are defined as (see [21]):

$$\begin{aligned} z \wedge z' &= (z_1 \wedge z'_1, \ldots, z_n \wedge z'_n), \\ \bar{z} &= (\bar{z}_1, \ldots, \bar{z}_n), \\ z \leq z' &\Leftrightarrow z_1 \leq z'_1, \ldots, z_n \leq z'_n, \\ z < z' &\Leftrightarrow z \leq z' \text{ and } z \neq z'. \end{aligned}$$

The definition of $\vee$ is analogous to that of $\wedge$. Additionally, we define the *population* of a binary vector

$$\rho : \{0,1\}^n \to \mathbb{N}^0$$

as the count of its 1 values.

*Department of Computer Science, University of Bremen, Germany, bgenov@informatik.uni-bremen.de

Extreme Ray Enumeration. For a polyhedral cone $\mathcal{P}$ the *extreme ray enumeration* is defined as the problem to find $\mathcal{V}^E$ out of A. This problem, which is identical to the *vertex enumeration* of polytopes, has a number of algorithmic solutions developed over the years. The first one, called the *double description method*, was introduced by Motzkin et al. [20] and later improved by Fukuda et al. [16]. Further algorithms with historical significance are the *algorithm of Chernikova* [12, 19], the *beneath-and-beyond method* of Seidel [23, 15], the *randomized algorithm* of Clarkson and Shor [13], the *derandomized algorithm* of Chazelle [11] and the *reverse search method* of Avis and Fukuda [3, 4]. For the moment, there is no general algorithm performing in time polynomial in the size of A and $\mathcal{V}^E$ [1, 2]. The question of whether such an algorithm exists is open as well [18]. Yet, for nondegenerate problems polynomial time solutions are available [24, 3, 10].

In this paper, we focus on the practical implementation of incremental cutting plane algorithms like the double description method and Chernikova's algorithm. Those start with an approximation cone $\mathcal{P}_1 \supseteq \mathcal{P}$ for which the extreme ray set $\mathcal{V}_1^E$ is known and perform a step by step refinement. At each step, the currently best known approximation $\mathcal{P}_i$ is cut with a new halfspace $\mathcal{H}_i = \{a_i^T \cdot x \geq 0\}$ which splits $\mathcal{V}_i^E$ into the subsets

$$\begin{aligned} \mathcal{V}_i^0 &= \{r^0 \in \mathcal{V}_i^E : a_i^T \cdot r^0 = 0\}, \\ \mathcal{V}_i^+ &= \{r^+ \in \mathcal{V}_i^E : a_i^T \cdot r^+ > 0\} \text{ and} \\ \mathcal{V}_i^- &= \{r^- \in \mathcal{V}_i^E : a_i^T \cdot r^- < 0\}. \end{aligned}$$

The set $\mathcal{V}_{i+1}^E$ contains $\mathcal{V}_i^0$, $\mathcal{V}_i^+$ and one new element for each pair of adjacent extreme rays $(r^+, r^-) \in (\mathcal{V}_i^+ \times \mathcal{V}_i^-)$. In practice, enumerating those pairs is the most time consuming part of the algorithm. We propose improvements related to the currently used data structures in order to speed up this process.

Assuming that $\mathcal{P}$ is pointed and thus $rank[A] = d$, the adjacency test of two extreme rays could be performed in two different ways known as a *combinational* (see Lemma 1) and an *algebraic test* (see Lemma 2). For the proof of both lemmas we refer to [16, Proposition 7]. Corollary 3 expresses an incomplete form of the algebraic test delivering either a negative or an indecisive result.

Lemma 1 (Combinational Test) *Two extreme rays $r', r'' \in \mathcal{V}^E$ are adjacent if and only if there is no other*

extreme ray $r''' \in \mathcal{V}^E$ *such that* $\psi(r') \wedge \psi(r'') < \psi(r''')$.

Lemma 2 (Algebraic Test) *Two extreme rays* $r', r'' \in \mathcal{V}^E$ *are adjacent if and only if* $rank[A'] = d - 2$ *where* $A' \in \mathbb{R}^{k \times d}$ *is a submatrix of* A *containing only those row vectors* a_i *for which* $z_i = 1$ *with* $z = \psi(r') \wedge \psi(r'')$.

Corollary 3 *Two extreme rays* $r', r'' \in \mathcal{V}^E$ *are nonadjacent if* $\rho(z) < d - 2$ *with* $z = \psi(r') \wedge \psi(r'')$.

Using the above criteria we can outline a simple algorithm to identify all adjacent extreme ray pairs in $(\mathcal{V}_i^+ \times \mathcal{V}_i^-)$. First, we eliminate all pairs satisfying Corollary 3. We call this a *narrowing phase* and all remaining pairs *feasible* ones. Second, we check each feasible pair against Lemma 1 for a definite result. We call this a *verification phase*.

With regard to the narrowing phase, the enumeration of all feasible pairs has a quadratic complexity in the worst case, as each pair may indeed be a feasible one. If, however, the feasible pairs are only a small fraction, the enumeration could be sped up by applying the *divide-and-conquer* approach. First, the set $\mathcal{V}_i^+$ is partitioned into finitely many subsets, and then for each $r^- \in \mathcal{V}_i^-$ the search for feasible pairs is limited to those subsets which can produce a valid result. A closely related problem in metric spaces is the *fixed-radius near neighbor search* [7, 8]. Note that we are dealing here with a nonmetric space.

The divide-and-conquer approach is also applicable in the verification phase. Each application of Lemma 1 is basically a *partial match query* [22] on $\mathcal{V}_i^E$ where the result is reduced to the existence or nonexistence of a ray r''' matching the given active set constraint. A general analysis on the lower bounds of this problem (referred to as a *no partial match* query) can be found in [9].

Contributions and Related Work. Thus far, the most efficient implementation of the outlined algorithm has been given by Terzer et al. [25, 26]. Terzer et al. introduced the *bit pattern tree* (here *bp-tree*), a data structure based on Bentley's *k-d tree* [6], on which near neighbor and partial match queries are performed. The overall performance of the implementation, however, depends very much on the structure of the bp-trees. Differently structured trees may require completely different number of operations to process the same set of queries. We made use of the fact that in our case all query inputs are known before the tree creation and developed an optimization called *query bits neutralization*. This method for tree creation considered the query inputs during the creation process. The so generated *bp-qbn-trees* accelerated the overall computation for most of the investigated problems. In some cases, the calculation time was reduced by more than 80%. Furthermore, we examined the performance of *vp-trees* (vantage point trees) [28], also known as *metric trees* [27], as alternative data structures for the algorithm. In this respect, we identified scenarios for which they tend to perform better than bp-trees.

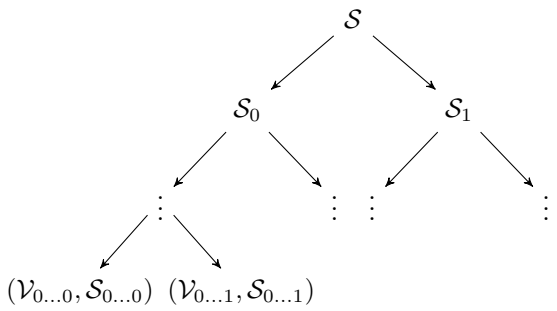

Figure 1: Generic structure of the binary tree.

2 Binary Trees for Adjacency Tests

In order to examine the performance of bp- und vp-trees we created a generic framework which supports adjacency tests using different binary tree types. In this section, we briefly introduce three major aspects of its functionality: tree generation, narrowing and verification. Functions whose implementation differs for different tree types are called *generic*. Specific implementations for bp- and vp-trees are presented in Sections 3 and 4.

Generation. The generation of a binary tree for some set of extreme rays $\mathcal{V}$ involves two major steps. First, $\mathcal{V}$ is recursively partitioned into finitely many subsets $\mathcal{V}_{(0|1)^*}$. Each subset corresponds to a single leaf node of the resulting tree. Second, to each tree node, no matter if intermediate or leaf, an auxiliary data $\mathcal{S}_{(0|1)^*}$ is attached. This data is produced during the generation process and encapsulates certain properties of the extreme rays in the subsequent leaf nodes. The basic structure of the resulting tree is shown in Figure 1. Its recursive creation is covered in Function 1 which returns either an intermediate node consisting of two subnodes or a leaf one if no further partitioning is desired. The generic function *partition* splits the set $\mathcal{V}$ into two disjoint subsets according to some criteria.

Narrowing. In the narrowing phase, we construct a binary tree for the set $\mathcal{V}_i^+$ and perform an operation similar to a fixed-radius near neighbor search for each $r^- \in \mathcal{V}_i^-$. The implementation of the search procedure is given in Function 2. It recursively traverses the tree and enumerates all extreme rays r' building a feasible pair with the given ray r. At each recursion step, the

Function 1 $\texttt{create}(\mathcal{V}, \mathcal{S})$

if $\texttt{leaf\_condition}(\mathcal{V}, \mathcal{S})$ **then**
  return $(\mathcal{V}, \mathcal{S})$
else
  $((\mathcal{V}_0, \mathcal{S}_0), (\mathcal{V}_1, \mathcal{S}_1)) \leftarrow partition(\mathcal{V})$
  $\mathcal{T}_0 \leftarrow \texttt{create}(\mathcal{V}_0, \mathcal{S}_0)$
  $\mathcal{T}_1 \leftarrow \texttt{create}(\mathcal{V}_1, \mathcal{S}_1)$
  return $(\mathcal{T}_0, \mathcal{T}_1, \mathcal{S})$
end if

Function 2 $\texttt{cand}(r, \mathcal{T})$

Require: $\mathcal{T} = (\mathcal{T}_0, \mathcal{T}_1, \mathcal{S})$ if $\mathcal{T}$ is an intermediate node or $\mathcal{T} = (\mathcal{V}, \mathcal{S})$ if $\mathcal{T}$ is a leaf one.
if $proceed\_enum(r, \mathcal{S})$ **then**
  if $\texttt{leaf}(\mathcal{T})$ **then**
    return $\{r' \in \mathcal{V} : \rho(\psi(r) \wedge \psi(r')) \geq d - 2\}$
  else
    return $\texttt{cand}(r, \mathcal{T}_0) \cup \texttt{cand}(r, \mathcal{T}_1)$
  end if
else
  return $\emptyset$
end if

auxiliary data $\mathcal{S}$ is used to check whether such rays can be found in the subsequent leaf nodes. If that is not the case, the current branch is abandoned. This decision is made by the generic function *proceed_enum*.

Verification. Here a binary tree is constructed for the set $\mathcal{V}_i^{deg} = \{r \in \mathcal{V}_i^E : \rho(\psi(r)) > d - 1\}$ and then traversed once for each feasible pair (r', r'') produced in the narrowing phase. If no partial match according to Lemma 1 is found then the corresponding rays are adjacent. The query implementation is given in Function 3 where *proceed_ver* is a generic function using the auxiliary data $\mathcal{S}$ in order to check whether a certain branch can produce a match.

3 Bit Pattern Trees

In this section, we give the implementation of all generic functions for bp-trees and present the query bits neutralization method.

Implementation. The partitioning process for bp-trees is given in Function 4. It selects a vector $q \in \{0,1\}^n$ with exactly one zero bit and groups all extreme rays r for which q is a valid over-approximation of $\psi(r)$ into $\mathcal{V}_0$ and all others into $\mathcal{V}_1$. For each of the resulting subsets $\mathcal{V}_k$, $k \in \{0,1\}$, an active set union $u_{\mathcal{V}_k}$ over all $\psi(r), r \in \mathcal{V}_k$, is generated and attached to the corresponding tree node as an auxiliary data. It represents an over-approximation of the active set for each extreme

Function 3 $\texttt{ver}(r', r'', \mathcal{T})$

Require: $\mathcal{T} = (\mathcal{T}_0, \mathcal{T}_1, \mathcal{S})$ if $\mathcal{T}$ is an intermediate node or $\mathcal{T} = (\mathcal{V}, \mathcal{S})$ if $\mathcal{T}$ is a leaf one.
$e \leftarrow \psi(r') \wedge \psi(r'')$
if $proceed\_ver(e, \mathcal{S})$ **then**
  if $\texttt{leaf}(\mathcal{T})$ **then**
    if $\exists r''' \in \mathcal{V} \setminus \{r', r''\} : e < \psi(r''')$ **then**
      return false
    else
      return true
    end if
  else
    return $\texttt{ver}(r', r'', \mathcal{T}_0)$ **and** $\texttt{ver}(r', r'', \mathcal{T}_1)$
  end if
else
  return true
end if

Function 4 $\texttt{partition}_{bpt}(\mathcal{V})$

Let $q \in \{0,1\}^n$ **and** $\rho(\bar{q}) = 1$
$\mathcal{V}_0 \leftarrow \{r \in \mathcal{V} : \psi(r) \leq q\}$; $\mathcal{V}_1 \leftarrow \mathcal{V} \setminus \mathcal{V}_0$
$u_{\mathcal{V}_0} \leftarrow \bigvee_{r \in \mathcal{V}_0} \psi(r)$; $u_{\mathcal{V}_1} \leftarrow \bigvee_{r \in \mathcal{V}_1} \psi(r)$
return $((\mathcal{V}_0, u_{\mathcal{V}_0}), (\mathcal{V}_1, u_{\mathcal{V}_1}))$

ray contained in one of the subsequent leaf nodes. Consequently, applying Corollary 3 or Lemma 1 on $u_{\mathcal{V}_k}$ can in some cases indicate the result for all extreme rays stored in the subsequent leaf nodes. The implementation of *proceed_enum* (see Function 5) and *proceed_ver* (see Function 6) for bp-trees rests on the above implication.

Function 5 $\texttt{proceed\_enum}_{bpt}(r, \mathcal{S})$

Require: $\mathcal{S} = u_{\mathcal{V}}$
if $\rho(\psi(r) \wedge u_{\mathcal{V}}) \geq d - 2$ **then**
  return true
else
  return false
end if

Query Bits Neutralization. Using the vector q, at each partitioning step we can influence all active set unions in the left branch by defining a specific position at which their value is zero. Consequently, we can use this fact to stimulate the elimination of branches from the search procedures in both phases. In the narrowing phase, for example, we can intendedly plant zeros on positions which are likely to meet nonzero ones in the query input $\psi(r)$. The idea is to neutralize those positions in $\psi(r)$ which are likely to be 1 and thus reduce the value of $\rho(\psi(r) \wedge u_{\mathcal{V}})$ as much as possible. We call the so chosen zero positions *neutralizers*. The determination of the

Function 6 $\texttt{proceed\_ver}_{bpt}(e, \mathcal{S})$

Require: $\mathcal{S} = u_{\mathcal{V}}$
  if $e < u_{\mathcal{V}}$ **then**
    return true
  else
    return false
  end if

neutralizers can be done at the beginning of the narrowing phase by analyzing the set of all query inputs. Thus, if the bp-tree is built out of $\mathcal{V}_i^+$ then each vector q is extracted out of $\mathcal{V}_i^-$.

Neutralizers are also applicable in the verification phase as each zero position in $u_{\mathcal{V}}$ which is nonzero in e leads to termination of the search in the current branch. In this phase, however, analyzing the input data may cause a substantial overhead due to its generally large size. For those cases, the sets $\mathcal{V}_i^+$ and $\mathcal{V}_i^-$ could be used to determine the neutralizers instead as $e \in (\mathcal{V}_i^+ \times \mathcal{V}_i^-)$.

There are two major metrics to evaluate the quality of the query bits neutralization. First, the probability of each neutralizer to meet a nonzero bit for some arbitrary query input. We call this a *hit probability*. Second, the number of neutralizers per active set union. It should be pointed out that those two factors may easily build a trade-off. A good neutralizer according to the first metric might also produce a bad partitioning where $|\mathcal{V}_0| << |\mathcal{V}_1|$ (see Function 4). In those cases, the impact of the neutralizer is considerably reduced as it will be planted only in a small fraction of the subsequent active set unions. Consequently, for an effective neutralizer selection both hit probability and potential partitioning should be taken into account.

4 Vantage Point Trees

In this section, we give the implementation of all generic functions for vp-trees.

In the *partition* function (see Function 7) we select an arbitrary extreme ray $v \in \mathcal{V}$, the so called *vantage point*, and measure the distance from v to all other rays in $\mathcal{V}$ using the distance function

$$\delta(v, r) = \rho(\psi(v)) - \rho(\psi(v) \wedge \psi(r)).$$

Let δ_{max} be the maximal measured distance. The set $\mathcal{V}$ is then split by selecting some arbitrary distance $l, 0 < l \leq \delta_{max}$, and grouping all rays r with $\delta(v, r) < l$ into $\mathcal{V}_0$ and all others into $\mathcal{V}_1$. To each of the resulting subsets $\mathcal{V}_k$, $k \in \{0, 1\}$, we attach as an auxiliary data the active set union $u_{\mathcal{V}_k}$, the vantage point v, the distance range $q_{\mathcal{V}_k}$ and the population range $p_{\mathcal{V}_k}$. The distance range $q_{\mathcal{V}_k}$ is a closed interval bounded by the minimal and maximal distance from v to any ray in $\mathcal{V}_k$. The population range $p_{\mathcal{V}_k}$ is a closed interval bounded

Function 7 $\texttt{partition}_{vpt}(\mathcal{V})$

  Let $v \in \mathcal{V}$
  $\delta_{max} \leftarrow \delta(v, r)$ with $r \in \mathcal{V} : \forall r' \in \mathcal{V}, \delta(v, r) \geq \delta(v, r')$
  Let $l \in (0, \delta_{max}]$
  $\mathcal{V}_0 \leftarrow \{r \in \mathcal{V} : \delta(v, r) < l\}$; $\mathcal{V}_1 \leftarrow \mathcal{V} \setminus \mathcal{V}_0$
  $u_{\mathcal{V}_0} \leftarrow \bigvee_{r \in \mathcal{V}_0} \psi(r)$; $u_{\mathcal{V}_1} \leftarrow \bigvee_{r \in \mathcal{V}_1} \psi(r)$
  $q_{\mathcal{V}_0} \leftarrow [0, l-1]$; $q_{\mathcal{V}_1} \leftarrow [l, \delta_{max}]$
  $p_{\mathcal{V}_0} \leftarrow [p_{min_0}, p_{max_0}]$ where
    $\forall r_0 \in \mathcal{V}_0, p_{min_0} \leq \rho(\psi(r_0)) \leq p_{max_0}$
  $p_{\mathcal{V}_1} \leftarrow [p_{min_1}, p_{max_1}]$ where
    $\forall r_1 \in \mathcal{V}_1, p_{min_1} \leq \rho(\psi(r_1)) \leq p_{max_1}$
  $\mathcal{S}_0 \leftarrow (u_{\mathcal{V}_0}, v, q_{\mathcal{V}_0}, p_{\mathcal{V}_0})$; $\mathcal{S}_1 \leftarrow (u_{\mathcal{V}_1}, v, q_{\mathcal{V}_1}, p_{\mathcal{V}_1})$
  return $((\mathcal{V}_0, \mathcal{S}_0), (\mathcal{V}_1, \mathcal{S}_1))$

by the minimal and maximal population of the elements in $\mathcal{V}_k$. With respect to the narrowing phase, the implementation of *proceed_enum* (see Function 8) rests on the implication given in the following Lemma 4.

Function 8 $\texttt{proceed\_enum}_{vpt}(r, \mathcal{S})$

Require: $\mathcal{S} = (u_{\mathcal{V}}, v, q_{\mathcal{V}}, p_{\mathcal{V}})$ with $q_{\mathcal{V}} = [q_{min}, q_{max}]$ and $p_{\mathcal{V}} = [p_{min}, p_{max}]$
  $c_1 \leftarrow \rho(\psi(r)) + \delta(v, r) - q_{min}$
  $c_2 \leftarrow p_{max} + q_{max} - \delta(v, r)$
  if $c_1 \geq d - 2$ **and** $c_2 \geq d - 2$ **then**
    return true
  else
    return false
  end if

Lemma 4 *If the extreme rays $r', r'' \in \mathcal{V}^E$ are adjacent then for any $v \in \mathcal{V}^E$*

$$\rho(\psi(r')) + \delta(v, r') - \delta(v, r'') \geq d - 2 \text{ and}$$
$$\rho(\psi(r'')) + \delta(v, r'') - \delta(v, r') \geq d - 2.$$

Let in the context of Lemma 4 r' be the argument r from Function 8. Then for r'' we use the intervals $q_{\mathcal{V}}$ and $p_{\mathcal{V}}$ as an over-approximation for the distance $\delta(v, r'')$ and the population $\rho(\psi(r''))$. As a consequence, the traversal of a certain tree branch needs to be proceeded only if the inequations given in Lemma 4 hold for any distance from $q_{\mathcal{V}}$ and any population from $p_{\mathcal{V}}$. Otherwise the nonexistence of feasible candidates in the subsequent leaf nodes is guaranteed.

For the implementation of *proceed_ver* (see Function 9) we apply the condition defined in Lemma 5. In a similar way, for the distance $\delta(v, r''')$ and the population $\rho(\psi(r'''))$ we use the ranges $q_{\mathcal{V}}$ and $p_{\mathcal{V}}$ from the auxiliary data. In order to maximize the branch elimination the conditional function for bp-trees is invoked as an additional criterion.

The proofs of Lemmas 4 and 5 can be found in the full version of the paper.

Lemma 5 *If $r', r'', r''' \in \mathcal{V}^E$ are extreme rays such that $e < \psi(r''')$ for $e = \psi(r') \wedge \psi(r'')$ then for any $v \in \mathcal{V}^E$*

$$\rho(\psi(v) \wedge e) \leq \rho(\psi(v)) - \delta(v, r''') \text{ and}$$
$$\rho(\overline{\psi(v)} \wedge e) \leq \rho(\psi(r''')) - \rho(\psi(v)) + \delta(v, r''').$$

Function 9 $\texttt{proceed\_ver}_{vpt}(e, \mathcal{S})$

Require: $\mathcal{S} = (u_\mathcal{V}, v, q_\mathcal{V}, p_\mathcal{V})$ with $q_\mathcal{V} = [q_{min}, q_{max}]$ and $p_\mathcal{V} = [p_{min}, p_{max}]$
  $c_1 \leftarrow \rho(\overline{\psi(v)} \wedge e) - \rho(\psi(v)) + q_{min}$
  $c_2 \leftarrow \rho(\overline{\psi(v)} \wedge e) + \rho(\psi(v)) - p_{max} - q_{max}$
  if ($c_1 \leq 0$ **and** $c_2 < 0$) **or** ($c_1 < 0$ **and** $c_2 \leq 0$) **then**
    return $\texttt{proceed\_ver}_{bpt}(e, u_\mathcal{V})$
  else
    return false
  end if

5 Results

In this section, we present the results from a small study comparing the performance of bp-trees, bt-qbn-trees and vp-trees. We used our own implementation of the double description method[1] in order to apply the vertex and facet enumeration on different polytopes. Those included samples from the work of Avis et al. [1, 2] and a small collection of cut [5], metric [14] and randomly generated 0/1 polytopes by polymake [17]. Table 1 summarizes the results for the cut polytope c_7, the metric one m_7, the product of cyclic polytopes $cyc_{4\_26\_2}$, the product of two simplices and a cube $glue_{54}$, the truncated polytope $trunc_{50}$ and three randomly generated 0/1 polytopes, one of which was joined with a hypercube.

	d	bp-trees	bp-qbn-trees	vp-trees
$cyc_{4\_26\_2}$	9	118.6	75.2	117.5
c_7	22	110.4	53.0	102.9
m_7	22	3463.8	1028.1	9393.9
rnd_{36}	22	1391.8	206.4	319.2
rnd_{cube}	31	756.3	715.2	137.9
$trunc_{50}$	51	470.0	582.7	421.0
$glue_{54}$	55	22.9	7.3	5.5
rnd_{64}	59	277.6	222.0	168.1

Table 1: Sum of narrowing and verification time (s)

On the basis of the experimental results, we outlined four major tendencies. First, vp-trees scaled best with growing dimensionality. This is visible, for instance, in the calculation times for rnd_{36} and rnd_{64}, which are similar problems in a different dimension. Second, bp-qbn-trees were not suitable for problems where only a small amount of data was processed at each step as the overhead for selecting neutralizers could not be compensated (see $trunc_{50}$). Third, the performance of vp-trees was very sensitive to the size of the population range $p_\mathcal{V}$ (see Function 8). The wider the range the worse the performance. Figure 2 illustrates the complexity of the narrowing phase for c_7 (snapshot).

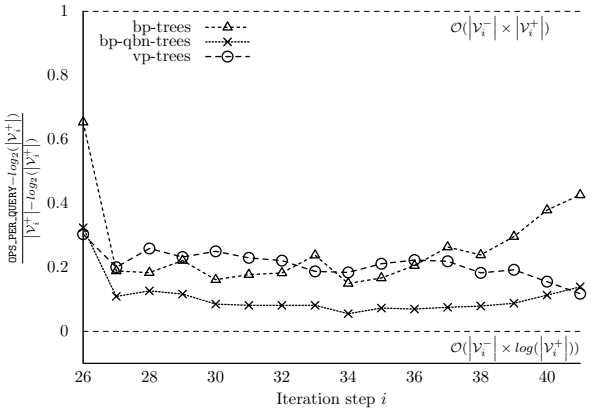

Figure 2: Narrowing phase for c_7 (snapshot).

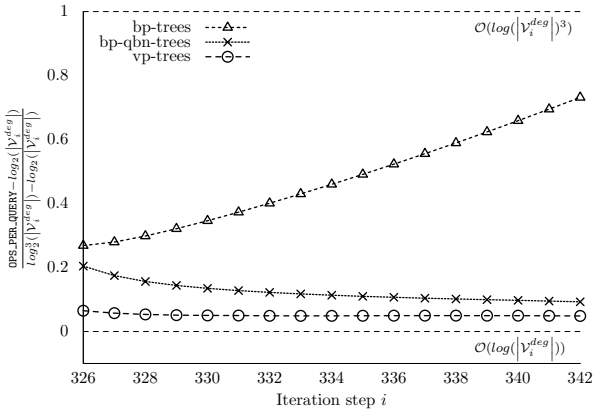

Figure 3: Verification phase for $glue_{54}$ (snapshot).

The execution of $\texttt{proceed\_ver}_{vpt}$ might have produced rowing phase for c_7. For the majority of steps vp-trees could not reach the efficiency of the bp-qbn-trees due to the size of the population range. In the final steps $p_\mathcal{V}$ narrowed down which eventually boosted the vp-trees performance. In comparison, the verification phase of $glue_{54}$ (see Figure 3) illustrates a scenario with minimal population ranges. Finally, vp-trees showed considerably better results for rnd_{cube} which is a problem with an extremely high rate of negative tests in the verification phase. In the most time consuming steps, more than 99.99% of the tests were negative. In comparison, for m_7 this rate remained between 95 and 99%.

It is worth mentioning that for the generation of the complexity charts the invocation of $\texttt{proceed\_enum}_{vpt|bpt}$ and $\texttt{proceed\_ver}_{bpt}$ counted as one operation. The

[1] Available at www.informatik.uni-bremen.de/agbs/bgenov

up to two operations due to the potential call to $\texttt{proceed\_ver}_{bpt}$.

6 Conclusion

In this paper, we revisited the application of bp-trees within incremental extreme ray enumeration algorithms and proposed a dynamic optimization of the trees with regard to the particular input problem. For most of the investigated problems a reduction in the overall calculation time was achieved. Furthermore, we examined the general suitability of vp-trees as an alternative data structure and presented problems for which vp-trees outperformed bp-trees. Still, further improvements are necessary so that vp-trees become competitive in the general case.

References

[1] D. Avis and D. Bremner. How good are convex hull algorithms? In *Proceedings of the eleventh annual symposium on Computational geometry*, SCG '95, pages 20–28, New York, USA, 1995. ACM.

[2] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms? *Computational Geometry: Theory and Applications*, 7:265–301, 1997.

[3] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. In *Proceedings of the seventh annual symposium on Computational geometry*, SCG '91, pages 98–104, New York, USA, 1991. ACM.

[4] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(13):21–46, 1996.

[5] F. Barahona and A. R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36(2):157–173, 1986.

[6] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[7] J. L. Bentley. A survey of techniques for fixed radius near neighbor searching. Technical report, Stanford, CA, USA, 1975.

[8] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

[9] A. Borodin, R. Ostrovsky, and Y. Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 312–321, New York, USA, 1999. ACM.

[10] D. Bremner, K. Fukuda, and A. Marzetta. Primal-dual methods for vertex and facet enumeration. *Discrete & Computational Geometry*, 20:333–357, 1998.

[11] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.

[12] N. V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of system of linear inequalities. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 5:228–233, 1965.

[13] K. L. Clarkson and P. W. Shor. Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental. In *Proceedings of the fourth annual symposium on Computational geometry*, SCG '88, pages 12–17, New York, 1988. ACM.

[14] A. Deza, K. Fukuda, D. Pasechnik, and M. Sato. On the skeleton of the metric polytope. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 125–136. Springer, 2001.

[15] H. Edelsbrunner. *Algorithms in combinatorial geometry*. Springer-Verlag, New York, 1987.

[16] K. Fukuda and A. Prodon. Double description method revisited. In *Combinatorics and Computer Science*. Springer-Verlag, Berlin/Heidelberg, 1996.

[17] E. Gawrilow and M. Joswig. polymake: a framework for analyzing convex polytopes. In *Polytopes — Combinatorics and Computation*. Birkhäuser, 2000.

[18] L. Khachiyan, E. Boros, K. Borys, K. M. Elbassioni, and V. Gurvich. Generating all vertices of a polyhedron is hard. *Discrete & Computational Geometry*, 39(1-3):174–190, 2008.

[19] H. Le Verge. A note on Chernikova's Algorithm. Technical Report 635, IRISA, Rennes, France, 1992.

[20] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. *The double description method, in Contributions to the Theory of Games*, volume II, pages 51–73. Princeton University Press, 1953.

[21] C. Posthoff and B. Steinbach. *Logic functions and equations: Binary models for computer science*. Springer, Dordrecht, The Netherlands, 2004.

[22] R. Rivest. Partial-match retrieval algorithms. *SIAM Journal on Computing*, 5(1):19–50, 1976.

[23] R. Seidel. A convex hull algorithm optimal for point sets in even dimensions. Technical report, Vancouver, Canada, 1981.

[24] R. Seidel. *Output-size sensitive algorithms for constructive problems in computational geometry*. PhD thesis, Ithaca, USA, 1987.

[25] M. Terzer and J. Stelling. Accelerating the computation of elementary modes using pattern trees. In *Proceedings of the 6th international conference on Algorithms in Bioinformatics*, WABI '06, pages 333–343, Berlin/Heidelberg, 2006. Springer-Verlag.

[26] M. Terzer and J. Stelling. Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics*, 24:2229–2235, 2008.

[27] J. K. Uhlmann. Metric trees. *Applied Mathematics Letters*, 4(5):61–62, 1991.

[28] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA '93, pages 311–321, Philadelphia, PA, USA, 1993. SIAM.