Efficient computation of squarefree separator polynomials

Michela Ceria[1], Teo Mora[2], and Andrea Visconti[1]

[1] Department of Computer Science, Università degli Studi di Milano, Italy
michela.ceria@gmail.com, andrea.visconti@unimi.it,

[2] Department of Mathematics, University of Genoa, Italy
theomora@disi.unige.it

Abstract. Given a finite set of distinct points, a separator family is a set of polynomials, each one corresponding to a point of the given set, such that each of them takes value one at the corresponding point, whereas it vanishes at any other point of the set. Separator polynomials are fundamental building blocks for polynomial interpolation and they can be employed in several practical applications. Ceria and Mora recently developed a new algorithm for squarefree separator polynomials. The algorithm employs as a tool the point trie structure, first defined by Felszeghy-Ráth-Rónyai in their Lex game algorithm, which gives a compact representation of the relations among the points' coordinates. In this paper, we propose a fast implementation in C of the aforementioned algorithm, based on an efficient storing and visiting of the point trie. We complete the implementation with tests on some sets of points, giving different configurations of the corresponding tries.

Keywords: Separator polynomials, point trie

1 Introduction

Given a finite set of distinct points $\mathbf{X} := \{P_1, ..., P_N\} \subset \mathbf{k}^n$, separator polynomials for $\mathbf{X}$ are polynomials $Q_1, ..., Q_N \in \mathbf{k}[x_1, ..., x_n]$ such that $\forall 1 \leq i, j \leq n$, $Q_i(P_j) = \delta_{i,j}$.

They have many applications in all fields of science, since they are the building blocks for polynomial interpolation. They are usually computed by means of some version Moeller algorithm [6,5], which gives also the whole Groebner basis for the ideal $I(\mathbf{X})$ of the points. The currently available implementations of Moeller algorithm have complexity $\mathcal{O}(n^2N^3)$ (see [7, Vol.2, 29.4.2]); if the improvement by Lundqvist would have been implemented (it is still not available) we would have complexity $\mathcal{O}(min(N, n)N^3 + nN^2)$. There are also some formulas to compute such polynomials [2,4], but, as remarked in [4], they add redundancy to the polynomials, which can be removed after computing them.

In [1], the authors developed an algorithm, based on Felszeghy-Ráth-Rónyai's point trie, which directly computes the separator polynomials, avoiding the redundancy so not needing to prune it afterwards. The complexity of the algorithm is $\mathcal{O}(N^2 log(N)n + N\min(N, nr))$.

The aim of this paper is to describe an efficient implementation of the algorithm in [1] which leans on an efficient storing and visiting of the point trie. We complete the implementation with tests on some sets of points, giving different configurations of the corresponding tries.

2 Notation

Throughout this paper we mainly follow the notation of [7]. We denote by $\mathcal{P} := \mathbf{k}[x_1, ..., x_n]$ the ring of polynomials in n variables with coefficients in the field $\mathbf{k}$.

Let $\mathbf{X} = \{P_1, ..., P_N\} \subset \mathbf{k}^n$ be a finite set of distinct points

$$P_i := (a_{1,i}, ..., a_{n,i}),\ i = 1, ..., N.$$

We call

$$I(\mathbf{X}) := \{f \in \mathcal{P} :\ f(P_i) = 0,\ \forall i\},$$

the *ideal of points* of $\mathbf{X}$.

Finally we recall some definitions from Graph Theory, following the notation of [2].

Definition 1 *We call* tree *a connected acyclic graph. A* rooted tree *is a tree where a special* vertex *(or* node*) called* root *is singled out.*

We say that a vertex is on the h-th *level* of the tree if its distance from the root is h, i.e. we have to walk on h edges to come from the root to the given vertex. If v is a vertex different from the root, and u is the vertex preceding v on the path from the root, then u is the *parent* of v and v is a *child* of u. Two vertices with the same parent are called *siblings*. If v is a vertex different from the root and u is on the path from v to the root, then u is an ancestor of v and v is a descendant of u. Clearly the root has no parent. We call *leaves* all the vertices having no children and we say that a *branch* is a path from the root to a leaf.

We consider always trees where all branches have the same length. The vertices lying in the last level of the tree coincide with the *leaves*; there are no vertices of the tree under them.

3 Separator polynomials

In this section, following the notation of [4], we define separator polynomials.

Definition 2 *A* family of separators *for a finite set of distinct points* $\mathbf{X} = \{P_1, ..., P_N\}$ *is a set* $Q = \{Q_1, ..., Q_N\}$ *s.t.* $Q_i(P_j) = \delta_{ij}$, $1 \leq i, j \leq N$, *where* δ_{ij} *denotes the Kronecker delta.*

Separators are useful building blocks for polynomial interpolation, in the sense that, every time one has to find a polynomial $p \in \mathbf{k}[x_1, .., x_n]$ such that $p(P_i) = b_i$ for $b_i \in \mathbf{k}$, $1 \leq i \leq N$, it is possible to find it by computing a separator family for $\mathbf{X}$ and setting

$$p(x_1, ..., x_n) = \sum_{i=1}^{N} b_i Q_i(x_1, ..., x_n).$$

We denote the points in $\mathbf{X}$, as in [1], so by $P_i := (a_{1,i}, ..., a_{n,i})$, $i = 1, ..., N$, and we define the witness matrix $C = (c_{i,j})$ [4], as the symmetric matrix s.t., for $i, j \in \{1, ..., N\}$, $c_{i,j} = 0$ if $i = j$ and if $i \neq j$, $c_{i,j} = \min\{h : 1 \leq h \leq n \text{ s.t. } a_{h,i} \neq a_{h,j}\}$. In other words, the witness matrix represents the minimal index h such that two points share the first $1, ..., h-1$ coordinates, but they have different h-coordinate. Using this matrix and the coordinates of the points, we can compute the polynomials we will use as constituting factors for our separator polynomials:

$$p_{i,j}^{[c_{i,j}]} = \frac{x_{c_{i,j}} - a_{c_{i,j},j}}{a_{c_{i,j},i} - a_{c_{i,j},j}}$$

In the paper [4], separator polynomials are built with the following variation of Lagrange's formula:

$$R_i = \prod_{i \neq j} \frac{x_{c_{i,j}} - a_{c_{i,j},j}}{a_{c_{i,j},i} - a_{c_{i,j},j}} = \prod_{j \neq i} p_{i,j}^{[c_{i,j}]}.$$

Then, it is observed that repeated factors do not affect the values taken by the polynomials R_i on the points of $\mathbf{X}$, and so the repeated factors are indicated as useless.

In the next section, we show how to compute *directly* the squarefree versions of these polynomials.

4 An algorithm for computing separator polynomials

In this section, following [1], we show how it is possible to compute directly squarefree separator polynomials, via a purely combinatorial algorithm. Our tool is the *point trie*, defined in [2] and presented in details also in [4].

Definition 3 *A* trie *is a rooted tree s.t. there is a symbol from a fixed alphabet, written on each edge.*

We use a trie of this kind, called *point trie*, to represent the points of the set $\mathbf{X}$ and the reciprocal relations among their coordinates. In particular, we label both the nodes and the edges:

- each edge is labelled by a coordinate; in particular the i-th coordinates are those labelling edges connecting nodes at levels $i - 1$ and i;

– the nodes, denoted by $v_{i,u}$, contain as label sets $V_{i,u}$ of indices, identifying the points whose $1...i$-th coordinates coincide (at least until level i). If for some i, u, $|V_{i,u}| \geq 2$ we call its elements *twin points*.

The trie is constructed iteratively on the points, appending to the trie the branches corresponding to the points one by one, as shown in the figure 1, that refers to the set $\mathbf{X} = \{P_1 = (1,0,0), P_2 = (0,1,0), P_3 = (1,1,2), P_4 = (1,0,3)\}$ of [1].

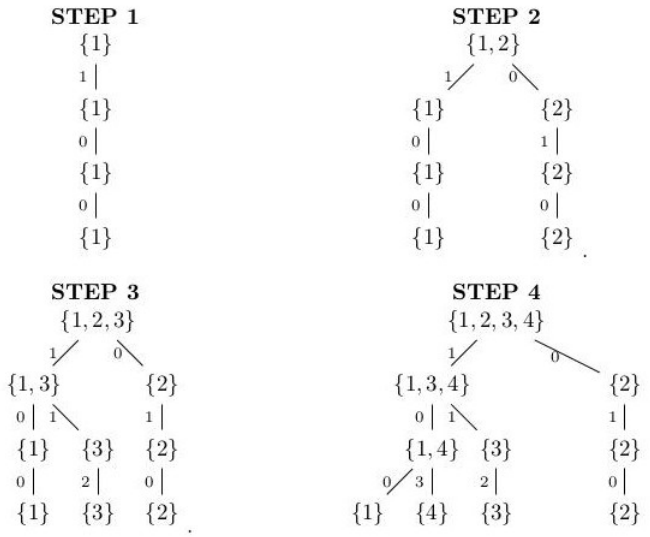

Fig. 1. The trie construction

Let now give a description of the algorithm, following [1]; in the next sections, we will give a closer look to the implementation.

If the given set $\mathbf{X}$ is composed by only one point $\mathbf{X} = \{P_1\}$, then the separator polynomial is $Q_1 = 1$. Suppose now to know the separator family $\{Q_1, ..., Q_{N-1}\}$ for $\{P_1, ..., P_{N-1}\}$ and to add the point P_N getting $\mathbf{X} = \{P_1, ..., P_N\}$. We compute the separator family $\{Q'_1, ..., Q'_N\}$ for $\mathbf{X}$, by computing the new polynomial Q_N, associated to P_N and by updating $Q_1, ..., Q_{N-1}$, making them fulfill definition 2 for the whole $\mathbf{X}$:

1. set $Q'_N = 1$;
2. $\forall j = 1, ..., n$ (the index j represent a level of the trie, i.e. a variable, so we are actually performing a pre-order walk on the trie), consider the (unique) node $v_{j,u}$ with $N \in V_{j,u}$.
3. $\forall v_{j,u'}$, sibling of $v_{j,u}$, pick some $\bar{i} \in V_{j,u'}$ and set $Q'_N = Q'_N p^{[j]}_{N,\bar{i}}$;

4. if, at level j, N has no twin points, i.e. if $|V_{j,u}| = 1$, then for each sibling $v_{j,u'}$, for each $i \in V_{j,u'}$, we set $Q'_i = Q_i p_{i,N}^{[j]}$.

Once concluded the above procedure, if, for some $1 \leq h \leq N$ a separator polynomial Q_h, has *not* been modified by the above steps, we set $Q'_h = Q_h$, getting a separator family $\{Q'_1, ..., Q'_N\}$ for $\mathbf{X} = \{P_1, ..., P_N\}$.

5 How to implement the algorithm

In this section, we give some concrete details on the implementation and we provide some results of our testing activities.

First of all, Figure 2 shows a toy example and represents both the trie construction and the resulting separator polynomials for the set $\mathbf{X} = \{P_1 = (0,0), P_2 = (1,2), P_3 = (4,2), P_4 = (1,3), P_5 = (7,4)\}$:

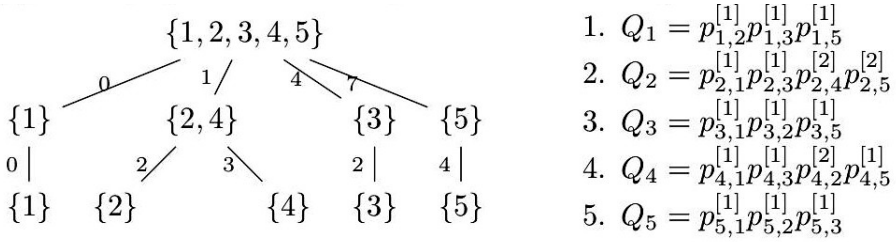

Fig. 2. A toy example: trie and separator polynomials

This trie is implemented in C using structs and pointers [8]. A graphical representation of its memory allocation is shown in figure 3

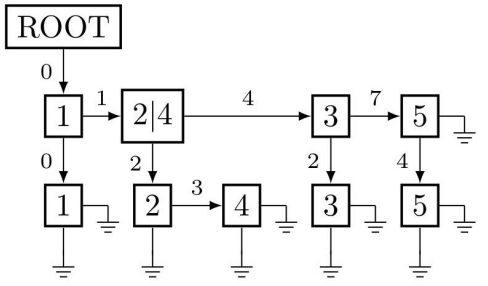

Fig. 3. Representation of the implemented trie for the toy example

We choose this approach because it minimizes the number of pointers allocated for each node and it also provides the possibility to add nodes dynamically at

runtime. Moreover, the approach adopted avoids to store useless nodes, keeping, for example, the twin nodes as a part of a single node. The advantages of this approach can be appreciated as soon as the number of points grows consistently.

Our testing activities has been executed on point sets over some fields of the form $\mathbf{k} = \mathbb{F}_{2^m}$, $m \in \mathbb{N}$. For simplicity, our implementation treats the elements of $\mathbb{F}_{2^m}$ as *positive integers*. In particular we

- fix a primitive element $\alpha \in \mathbb{F}_{2^m}$;
- set $\mathbb{F}_{2^m} = \{0, \alpha, \alpha^2, ..., \alpha^{2^m-1} = 1\}$;
- identify α^i with i.

Notice that the element 0 is actually identified with 0, whereas since $1 = \alpha^0 = \alpha^{2^m-1}$, we associate $2^m - 1$ to the element $1 \in \mathbb{F}_{2^m}$.

In order to evaluate the performance of the algorithm, we run our code on a laptop equipped with an Intel Core i7-7700HQ processor — cache 6MB, base frequency 2.8GHz, maximum frequency 3.8 GHz, 4 cores, 8 threads — and 32 GB of RAM. The operative system installed is Kubuntu 16.04.

In our testing activities, we generate points with three and four coordinates, which give different configurations of the trie, but it is trivial to extend it to more coordinates. The computational time spent to construct the trie and to compute the separator polynomials runs between 0.01 sec. — best case, 1,024 points, three coordinates — and 6 min. — worse case, 65,536 points, four coordinates (see table 1 for more details).

Table 1. Time spent to compute the separator polynomials

Number of points	Number of coordinates	Time spent (seconds)
1,024	3	0.01
4,096	3	1.02
12,341	3	2.68
16,384	3	8.41
65,536	3	240.52
512	4	0.01
65,536	4	373.37

6 Conclusions

In this paper, after recalling the definition an the importance of separator polynomials for interpolation, we have shown how to implement the algorithm introduced in [1] for computing them directly in a squarefree and redundancy-free way, which does not require pruning useless multiplicative factors.

Our testing activities suggest that the implementation of the algorithm does not use a large amount of memory and it runs quite fast enough, providing us the

possibility to run the code with a high number of nodes that is relevant w.r.t. the numbers found in literature — see [3] for example. Notice that in [3] the number of variables is bigger than the number of points, but this is not relevant for the resources and time employed by our implementation. Anyway, it is possible to improve the performances of our implementation by keeping track of the last sibling for each node and of the last twin stored in a node. Of course, at a cost of increasing memory consumption, one can speed up the code, since insertion of new points would not require reading all the siblings/twins anymore.

References

1. Ceria, M., Mora, T.: Combinatorics of ideals of points: a cerlienco-mureddu-like approach for an iterative lex game. preprint (2018)
2. Felszeghy, B., Ráth, B., Rónyai, L.: The lex game and some applications. Journal of Symbolic Computation 41(6), 663–681 (2006)
3. Laubenbacher, R., Stigler, B.: A computational algebra approach to the reverse engineering of gene regulatory networks. Journal of theoretical biology 229(4), 523–537 (2004)
4. Lundqvist, S.: Vector space bases associated to vanishing ideals of points. Journal of Pure and Applied Algebra 214(4), 309–321 (2010)
5. Marinari, M.G., Moeller, H.M., Mora, T.: Gröbner bases of ideals defined by functionals with an application to ideals of projective points. Applicable Algebra in Engineering, Communication and Computing 4(2), 103–145 (1993)
6. Möller, H.M., Buchberger, B.: The construction of multivariate polynomials with preassigned zeros. In: European Computer Algebra Conference. pp. 24–31. Springer (1982)
7. Mora, T.: Solving polynomial equation systems, 4 Vols., I (2003), II (2005), III (2015), IV (2016). Cambridge University Press
8. Ritchie, D.M., Kernighan, B.W., Lesk, M.E.: The C programming language. Prentice Hall Englewood Cliffs (1988)