BSP Application in Absolute Irreducibility Testing of Polynomials

Fatima Abu Salem

Computer Science Department, American University of Beirut

July 2006

Symbolic Computing

- Symbolic computing is concerned with the representation and manipulation of information in symbolic form.
- Computer algebra gaining more importance in many applied fields:
  - Cryptography
  - Coding Theory
  - Quantum Information and Computation
  - Computational Topology and Geometry
  - Many others...

High Performance in Computer Algebra

- Computer Algebra Systems are software that handle symbolic/exact computation.
- Major concern in devising such systems: the complex data structures and operations on them that should simulate a symbolic computation and deliver exact results.
- Many systems already exist that extend the feasibility of many symbolic algorithms on large problem sizes.
- Yet, much work is still needed to address many more algorithms that remain untackled from a “high-performance computational perspective”.

High Performance Computing in Computer Algebra

- Mathematical applications being the drive behind many computer algebra algorithms, versus physical science and engineering applications behind most numerical algorithms.
- Consequence: High-performance numerical libraries historically much more advanced than symbolic ones.
- Reasons?

Reasons? A Personal Perspective

- Perhaps mathematicians traditionally reluctant to address implementation aspects of algorithms.

Reasons? A Personal Perspective

- Perhaps mathematicians traditionally reluctant to address implementation aspects of algorithms.
- Engineers and physical scientists traditionally interested in stretching the limits of “feasible” computing, as dictated by their real-life applications

“Real-life” Algorithm Design

- Many factors affect the performance of an algorithm: Improving theoretical asymptotic performance is not enough.
- Of growing importance is an algorithm’s memory performance: how well does the algorithm (sequential or parallel) take advantage of the memory hierarchy?
- Compilers are far from achieving optimal performance in that respect.
- From a software engineering point of view, it is also extremely difficult and expensive to tune even the simplest of libraries to run on different platforms.

Automated Techniques

- Recent implementations of scientific (numerical) libraries have been endorsing automated techniques for optimisation.
- Similar techniques in the symbolic area are still at a very primitive stage, particularly because of the irregular data structures.

Cache Oblivious Algorithms (M. Frigo et al., 1999)

- *Automation*: Use a memory hierarchy effectively without knowledge of the underlying parameters such as the number of cache levels and their sizes.
- *Cache Complexity*: Incur the minimum number of cache misses within each level of a memory hierarchy.
- *Portability*: Obtain high performance across machines with different memory systems.

Open Question...

- When does a cache oblivious (CO) algorithm outperform a cache aware (CA) algorithm?

Open Question...

- When does a cache oblivious (CO) algorithm outperform a cache aware (CA) algorithm?

Current beliefs (?):

- Use a CO approach on a block-recursive algorithm.
- Use a CA approach on an iterative algorithm.

Once a BSP enthusiast...

- BSP attraction: Locality of run-time analysis, plus communication and synchronisation analysis in terms of the input size, provide an easy model through which to express the total running time.
- Brings about locality of spatial analysis.
- But not considering memory performance: good data locality is essential, not only a good load balance and a low communication/synchronisation overhead.

BSP in “real-life” implementations: Cache Oblivious BSP algorithms?

- (M. Frigo et al 2006): Cache oblivious multithreaded algorithms using Cilk.
- A strict assumption: non-interfering caches, in an asynchronous model, using a work-stealing scheduler.
- Outcome: fully proven mechanism to calculate the cache complexity of a multithreaded algorithm.
- Applications include block recursive matrix-matrix multiplication, stencil computations.

Open Question...

- Can BSP adapt to this, where message passing and synchrony are at the core?

BSP in “real-life” implementations: Cache Oblivious BSP algorithms?

- (D. Wise et al. 2006) Cache oblivious algorithms (via Morton-order indexing of matrices) using message passing paradigms.
- Applications include block recursive matrix operations and linear solvers.
- Achieve both parallel scalability and optimal cache performance.

Open Question...

- Can BSP adapt to this, combining mechanisms for calculating computational, communication, latency and cache complexities?

Once a BSP enthusiast...

- From a computational scientist perspective: different mathematical problems may still share many similar computational structures: instruction and data levels.
- Here is an example...

Polyhedral Methods

- Polyhedral methods have been used successfully in a number of symbolic algorithms:
  - Factorisation of bivariate polynomials over finite fields.
  - Absolute irreducibility testing of polynomials.
  - Homotopy Continuation methods for solving polynomial systems.
  - Many others...

Absolute Irreducibility Testing Via Polytopes

- Attraction of the polyderal approach: the fact that the non-zero coefficients of the input polynomial do not matter in the testing process makes it possible to show absolute irreducibility of families of polynomials, rather than single polynomials.

Background

- Let $\mathbb{F}[X_1, X_2, \ldots, X_n]$ be the ring of polynomials in n variables over an arbitrary field $\mathbb{F}$.
- For any vector $e = (e_1, \ldots, e_n)$ of non-negative integers, define $X^e := X_1^{e_1} \ldots X_n^{e_n}$. Let $f \in \mathbb{F}[X_1, \ldots, X_n]$ be given by

$$f := \sum_e a_e X^e \tag{1}$$

  where the sum is over finitely many points e in $\mathbb{N}^n$ called support vectors of f, and $a_e \in \mathbb{F}$.

- The Newton polytope of f is the polytope in $\mathbb{R}^n$ obtained as the convex hull of all exponents e for which the corresponding coefficient a_e is non-zero.

- It has integer vertices, since all the e are integral points. We call such polytopes *integral*.

- Given two polytopes Q and R, their *Minkowski sum* is defined to be the set

$$Q + R := \{q + r \,|\, q \in Q, r \in R\} \tag{2}$$

- When Q and R are integral polytopes, so is $Q + R$. If we can write an integral polytope P as a Minkowski sum $Q + R$ for integral polytopes Q and R then we call this an *integral decomposition*.

- The decomposition is *trivial* if Q or R has only one point, and P is *integrally decomposable* if it has at least one non-trivial decomposition.

- If a polytope has no non-trivial decompositions then it is *integrally indecomposable*.

- The Minkowski sum of two convex polytopes is also a convex polytope. A polytope of dimension 2 is a *polygon*, where the only proper faces are edges and vertices.

Strong Irreducibility Criterion

- We call f *absolutely irreducible* over $\mathbb{F}$ if it has no non-trivial factors over $\overline{\mathbb{F}}$, the algebraic closure of $\mathbb{F}$.
- Absolute irreducibility forms a strong irreducibility criterion because f has no irreducible factors over $\mathbb{F}$ if f is absolutely irreducible.

Theorem 1. *(Gao, 2001) Let $f \in \mathbb{F}[X_1, ..., X_n]$ with f not divisible by any non-constant X_i, for $1 \leq i \leq n$. If $\mathrm{Newt}(f)$ is integrally indecomposable, then f is absolutely irreducible.*

Testing indecomposability of polytopes

- Assuming that the polytope is given as a list of its vertices, the input size of this problem consists in the length of the binary representation of the coordinates of the vertices.
- (Gao and Lauder, 2001): deciding polygon indecomposability (and hence indecomposability of higher dimensional polytopes) is NP-complete
- Open problem: develop an efficient, polynomial time, deterministic or even randomised algorithm for testing general integral polytopes for indecomposability.

The Sequential Algorithm

- Let P be a convex polygon in $\mathbb{R}^2$, and let $v_0, ..., v_{m-1}$ denote its vertices ordered cyclically in a counter-clockwise direction.
- The edges of P are vectors of the form $E_i = v_{i+1} - v_i = (a_i, b_i)$, for $0 \leq i \leq m-1$, where $a_i, b_i \in \mathbb{Z}$ and the indices are taken modulo m.
- If $n_i = \gcd(a_i, b_i)$ and $e_i = (a_i/n_i, b_i/n_i)$, then $E_i = n_i e_i$, for $0 \leq i \leq m-1$.
- The sequence of vectors $\{n_i e_i\}_{0 \leq i \leq m-1}$ is called the edge sequence or polygonal sequence and uniquely identifies the polygon up to translation determined by v_0.

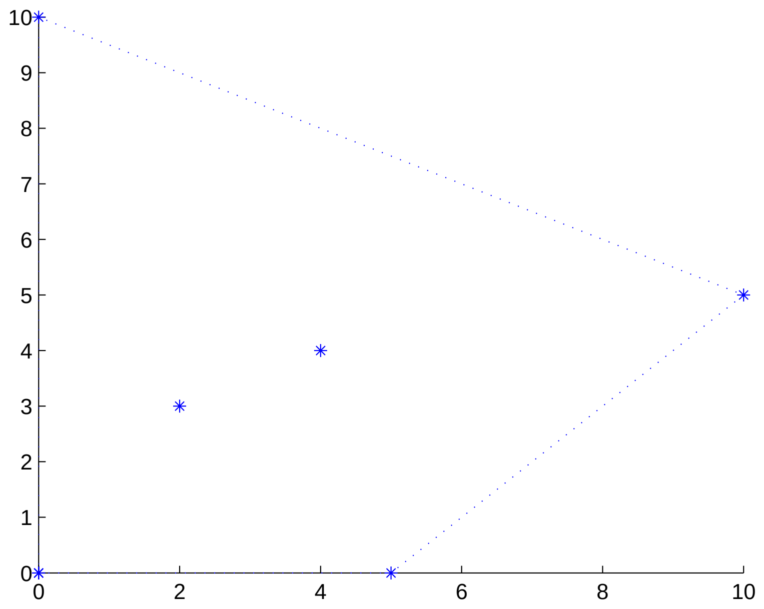
The polygon indecomposability testing algorithm

Algorithm 1. *(Gao and Lauder, 2001) Input: The edge sequence* $\{n_i e_i\}_{0 \leq i \leq m-1}$ *of an integral convex polygon* P *starting at a vertex* v_0 *where* $e_i \in \mathbb{Z}^2$ *are primitive vectors.*
Output: Whether P *is decomposable.*

Step 1: Compute the set IP *of all the integral points in* P*, and set* $A_{-1} = \emptyset$*.*

Example

$$f(x) = x^{10}y^5 + x^5 + x^4y^4 + x^2y^3 + y^{10}$$



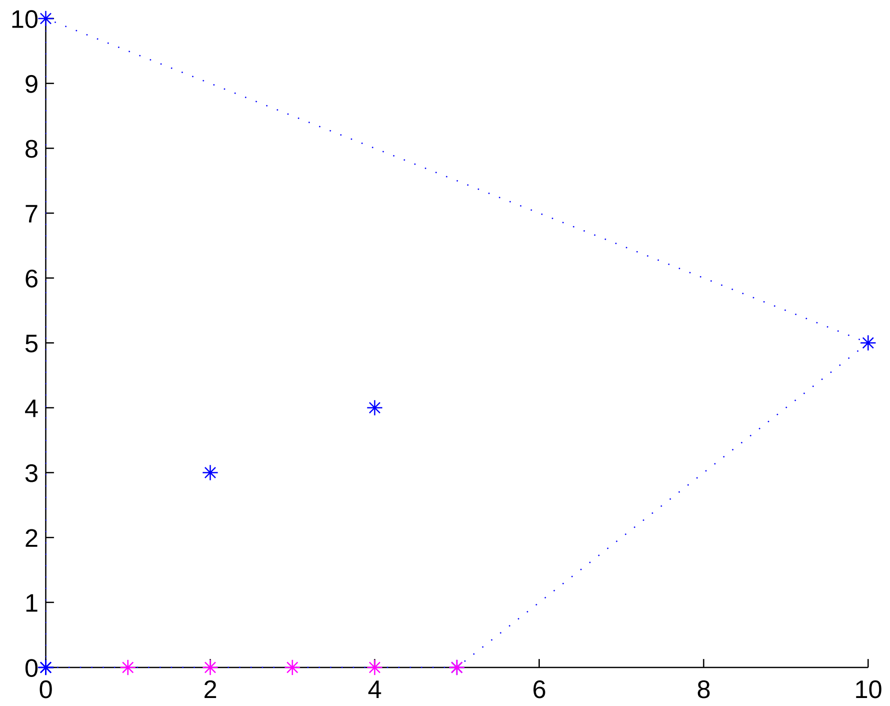
Step 2

For $i = 0, ..., m-2$, compute the set of points in IP that are reachable via the vectors $e_0, ..., e_i$:

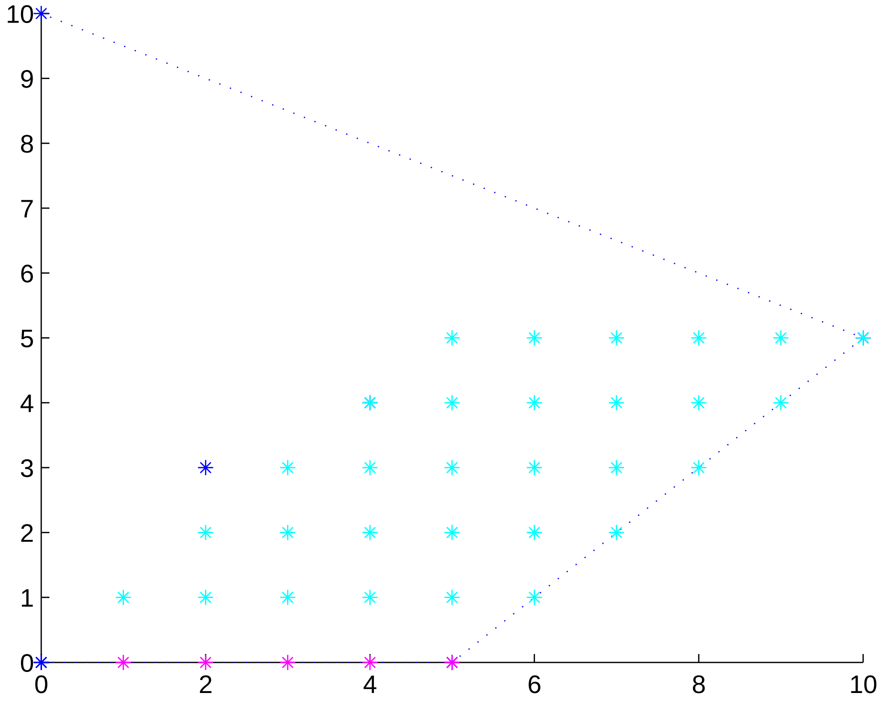
2.1: For each $k = 1, ..., n_i$, if $v_0 + ke_i \in IP$, then add it to A_i.

2.2: For each $u \in A_{i-1}$ and $k = 0, ..., n_i$, if $u + ke_i \in IP$, then add it to A_i.

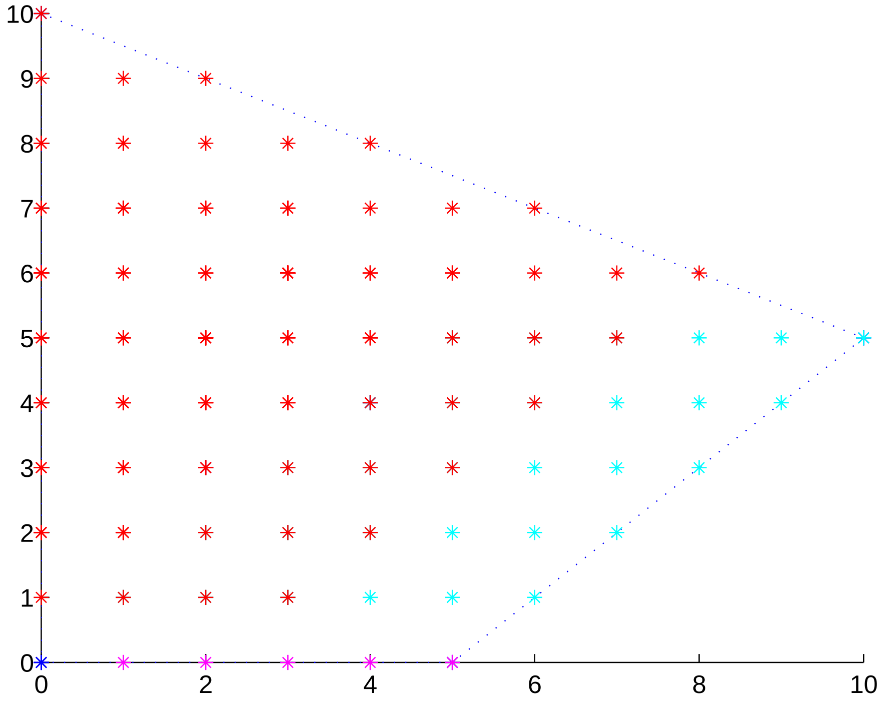
Example: First Iteration



Example: Second Iteration



Example: Third Iteration



Step 3

Step 3: Compute the last set A_{m-1}: For each $u \in A_{m-2}$ and $k = 0, ..., n_{m-1} - 1$, if $u + ke_{m-1} \in IP$ and $u + ke_{m-1}$ is not already in A_{m-1}, then add it to A_{m-1}.

(Similar to step 2.2, but using the last edge)

Step 4

Step 4: Return "Decomposable" if $v_0 \in A_{m-1}$ *and "Indecomposable" otherwise.*

Pseudo-polynomial run time

- (Gao and Lauder, 2001): a pseudo-polynomial time algorithm with a run-time complexity that is polynomial in the lengths of the sides of the polygon, rather than in the logarithm of the lengths.

Theorem 2. *(Gao and Lauder, 2001) The above algorithm decides decomposablitiy correctly in $O(t'mN)$ vector operations where t' is the number of integral points in P, m is the number of its edges, and N is the maximum number of integral points on an edge.*

Bottelneck

- The highest exponents in both x and y determine an upper bound on the length and width of the smallest rectangle enclosing the Newton polytope.

- Asymptotically, this determines a bound on the number of lattice (integral) points belonging to the Newton polytope.

Bottleneck

- Needed: a matrix data structure to store lattice points in the Newton polytope, and a corresponding data distribution in parallel which allows for an optimal load balance.

- Bottleneck: for asymptotically large degree polynomials, the spatial complexity of the sequential algorithm grows quadratically in the maximum of degrees in x and/or y.

- Reminiscent of BSP applications in parallelising various operations on matrices.

Detecting independent computations

- The inner-most computations of the iterative loop of Algorithm 1 can themselves be independent, generating a “horizontal” inherent parallelism across each iteration of the main loop over edges of the input polygon.

Lemma 1. *The above algorithm for polygon indecomposability testing contains two patterns of computations, one which describes a sequence of inter-dependent iterative steps for constructing new subsets of IP using previous subsets, and another pattern describing completely independent tasks for vector operations across a fixed iterative step.*

Proof

During a fixed stage $i = 0, ..., m - 2$ of the main loop iteration (Step 2 of Algorithm 1), one finds all points $u' \in A_i$ satisfying the following:

- $u' = v_0 + ke_i$, for $0 < k \leq n_i$, in which case the computations over all k require no information from the previous loop iteration of index $i - 1$, and each computation per fixed k requires no information apart from e_i.

This iterative step can be parallelised. A communication is needed when a processor's updates using vector translations produce lattice points which should belong to another processor. No synchronisation is yet needed.

Proof Cont'd

- $u' = u + ke_i$, for $0 \leq k \leq n_i$ and for all $u \in A_{i-1}$, in which case the computations over all u require information from the previous loop iteration of index $i-1$, each computation per fixed u and over all $k = 0, ..., n_i$ requires no information apart from n_i and e_i, and each computation per fixed u and fixed k requires no information apart from e_i.

Each processor operates only on the set of lattice points delivered to it in the previous iteration. It executes a full loop of vector translations applied to this set of lattice points. A communication is needed when a processor's updates using vector translations produce lattice points which should belong to another processor. No synchronisation is yet needed.

Proof Cont’d

At the end of every iteration looping over edges of the Newton polytope, a synchronisation is needed to make sure all broadcast lattice points have been delivered to their assigned processors.

Constructing a balanced load scheme

- A slight modification of Algorithm 1 to produce sets of points B_i in $\mathbb{R}^2$, for $0 \leq i \leq m-1$, constructed as follows:

  1. Initialise $B_i \leftarrow \{v_0\}$, for $i = -1, ..., m-1$.
  2. For $i = 0, ..., m-1$, compute the set of points of the plane that are reachable from v_0 via the vectors $e_0, ..., e_i$, and store them in B_i: For each $u \in B_{i-1}$ and $k = 0, ..., n_i$, add $u + ke_i$ to B_i.

How different are these sets?

- The origin v_0 is in every single set B_i

- Each B_i contains points reachable via $e_0, ..., e_i$ which are not necessarily in IP,

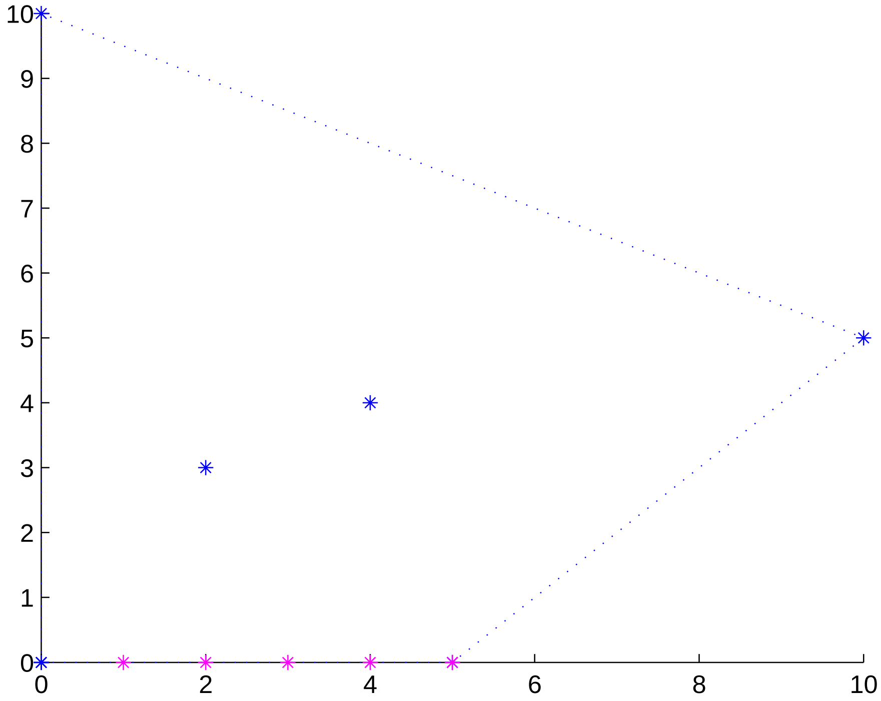
- The points $v_0 + ke_{m-1}$, for $k = 0, ..., n_{m-1}$, do lie in B_{m-1}.

- $A_i \subseteq B_i$ for every i. $\mathrm{Newt}(f) \subseteq B_{m-1}$ by construction of B_{m-1}.

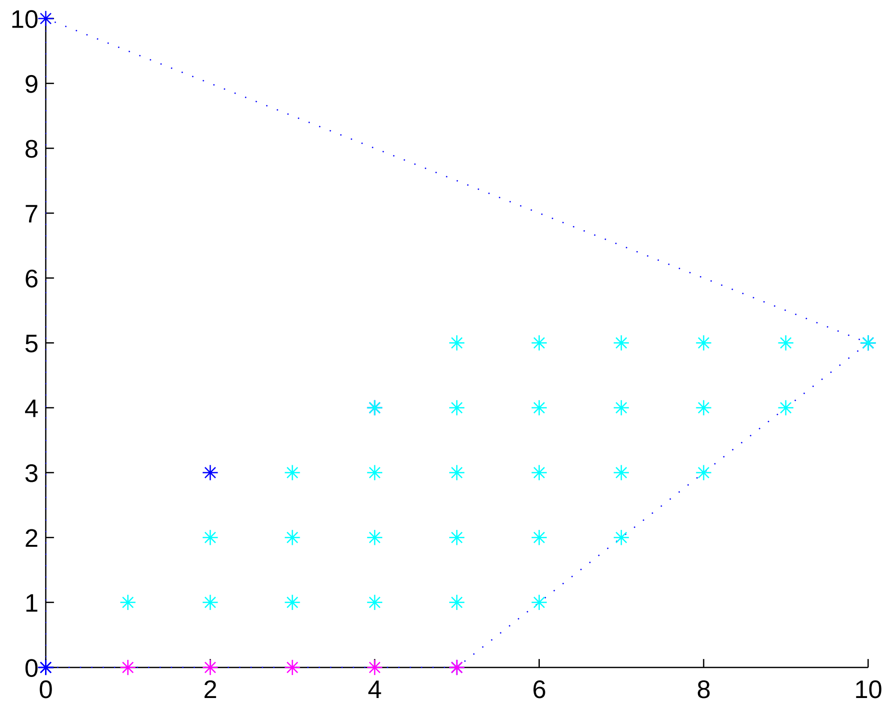
Why the sets B_i?

- The sets B_i have weaker conditions characterising their points, so that it will be simpler to describe the geometric pattern they follow.
- Since $A_i \subseteq B_i$ for every i, any such pattern will apply to elements of A_i.
- Asymptotically, and assuming a worst-case analysis, the sets A_i and B_i grow alike.

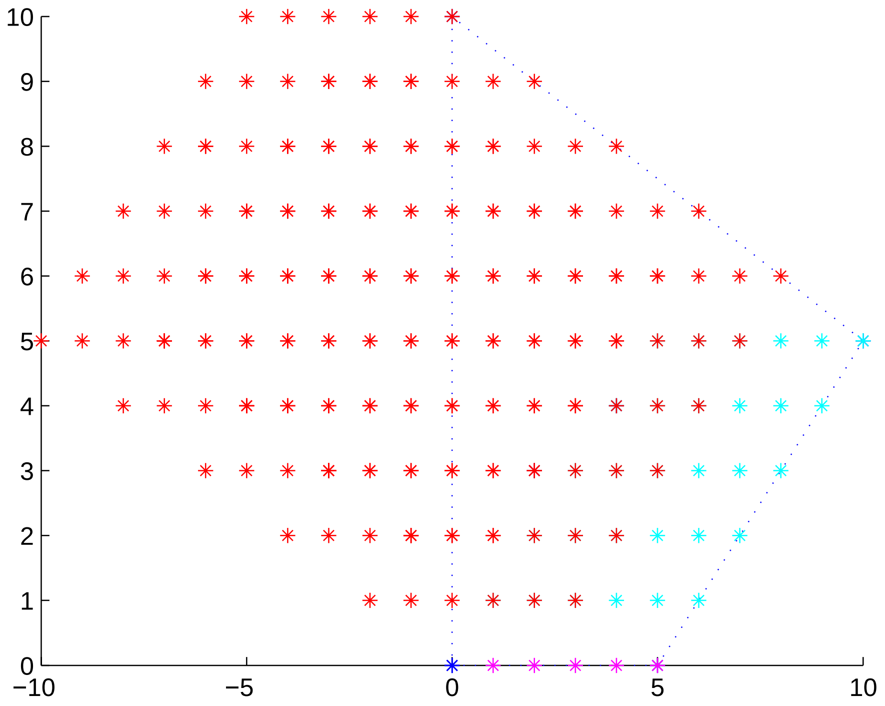
Example: First Iteration



Example: Second Iteration



Example: Third Iteration



Constructing a balanced data distribution

- The bulk of the work during any iterative step of Algorithm 1 takes place in well-defined active zones of the plane.
- One should thus avoid any form of data distribution where the polygon is triangulated into zones and each zone is exclusively assigned to one single processor.
- Specifically, this risks having some processors completely idle when others are engaged in the active zones.

Load Balancing

Lemma 2. *Let B_i, for $0 \leq i \leq m-1$, denote an active region of the plane as defined above, and let b_i denote the total number of lattice points belonging to the smallest square containing B_i. Let n_p denote the total number of processors operating in parallel such that $1 \leq n_p < \sqrt{b_i}$. The data distribution of integral points in IP which allocates every point $(k, k') \in B_i$ to the processor with identification number $id \equiv (k + k') \bmod n_p$ allows for a balanced load scheme, and assigns to each processor $O(b_i)/n_p$ integral points, where b_i represents an upper bound on the number of integral points in B_i.*

Note...

- In practice, and even though the condition $n_p < \sqrt{b_0}$ may not easily hold (as B_0 is simply the first edge of $\mathrm{Newt}(f)$, in which case b_0 denotes the number of integral points on that edge), the sizes of the sets B_i, for $i > 0$, start growing fast immediately afterwards

Data Distribution

Lemma 3. *Let x_{min}, y_{min}, x_{max} and y_{max} denote respectively the lowest x and y coordinates and the highest x and y coordinates appearing in any point belonging to $\mathrm{Newt}(f)$, and write*

$$\gamma = \max(y_{max} - y_{min}, x_{max} - x_{min}) \tag{3}$$

Let n_p denote the total number of processors operating in parallel such that $1 \leq n_p < 2\gamma$. The data distribution of integral points in IP which allocates every point $(k, k') \in \mathrm{Newt}(f)$ to the processor with identification number $id \equiv (k + k') \bmod n_p$ assigns to each processor $O(\gamma^2)/n_p$ integral points, where γ^2 is an upper bound on the number of lattice points in $\mathrm{Newt}(f)$.

The things we can do with BSP: BSP cost

Theorem 3. *Given a bivariate polynomial f over $\mathbb{F}$ of degree q with no non-constant monomial factors and with c nonzero terms, absolute irreducibility testing can be performed in parallel using*

$$\frac{O(cq^3)}{n_p} + \left(\frac{O(cq^3)}{n_p} + O(q+n_p)\right) g(n_p) + O(c)\ell(n_p) \qquad (4)$$

flops and $\frac{O(q^2)}{n_p}$ bits of storage, assuming q fits in a machine word, and

$$1 \leq n_p = O(q) \qquad (5)$$

The things we can do with BSP: Predict scalability

Corollary 1. *The BSP algorithm achieves efficiency $E_p \geq 1/2$ under the conditions*

1. $g(n_p) = O(1)$, *(see note below)*

2. $n_p < \gamma m N$,

3. $n_p < \gamma \left(\frac{\gamma m N}{n_p} - 1 \right)$,

4. $n_p \ell(n_p) = O(\gamma^2 m N)$.

Note...

- Although the first condition poses a heavy requirement on the communication parameter, the experimental results obtained benefit mainly from the low synchronisation cost, which is dependent on the number of edges of the input polygon.
- In the case of sparse polynomials, this number is usually very small, hence the speed-up we report in our experiments.
- We view our communication cost as an affordable requirement in practice, especially that the parallel algorithm promises absolute irreducibility testing of polynomials with significantly higher degrees than can be tested using a sequential version of the polygon decomposability testing algorithm.

Results

- Empirical results in the bivariate case reflect overall efficiency under reasonable parametric conditions that are implied by our theoretical analysis of the algorithm, and by significantly higher degree absolute irreducibility testing up to degree 30000.

- An extension of the bivariate algorithm yields a multivariate polynomial irreducibility test. We managed to test absolute irreducibility of trivariate polynomials with degree up to 30000 and of low degree multivariate polynomials with up to 3000 variables.

Results

- The mutlivariate algorithm, though heuristic in some parts, not only serves as a fast pre-test before invoking other well known algorithms, but also achieves absolute irreducibility testing for degrees which could not have been tackled otherwise: To the best of our knowledge, the degrees and the number of variables tackled in this work are a current world record.

Special Thanks

to Rob Bisseling:

For making me believe I could still do it from such a distance and at such hard times.

Special Thanks

to Richard Brent:

For having always restored my faith in humanity – despite the siege, the bombardments, the looming death – by simply being himself...