Using bipartite and multidimensional matching to select the roots of a system of polynomial equations

H. Bekker*, E. P. Braad*, B. Goldengorin**

(Department of Mathematics and Computing Science*, Faculty of Economic Sciences**), University of Groningen, P.O.Box 800, 9700 AV Groningen, The Netherlands. Email: bekker@cs.rug.nl, e.p.braad@wing.rug.nl, b.goldengorin@eco.rug.nl

Abstract

Assume that the system of two polynomial equations $f(x, y) = 0$ and $g(x, y) = 0$ has a finite number of solutions. Then the solution consists of pairs of an x-value and an y-value. In some cases conventional methods to calculate these solutions give incorrect results and are complicated to implement due to possible degeneracies and multiple roots in intermediate results. We propose and test a two-step method to avoid these complications. First all x-roots and all y-roots are calculated independently. Taking the multiplicity of the roots into account, the number of x-roots equals the number of y-roots. In the second step the x-roots and y-roots are matched by constructing a weighted bipartite graph, where the x-roots and the y-roots are the nodes of the graph, and the errors are the weights. Of this graph the minimum weight perfect matching is computed. By using a multidimensional matching method this principle may be generalized to more than two equations.

Key words: combinatorial optimization, bipartite matching, system of polynomial equations

1 Introduction

Consider a system of two polynomial equations

$$f(x, y) = 0 \quad g(x, y) = 0 \tag{1}$$

with numerical constants, and of low degree say ≤ 6. Assume that of this system it is known that the number of solutions in $\mathbb{C}$ is finite. Solving this system, using floating point arithmetic, is no problem for a computer algebra system, say MAPLE©. In general the calculations consist of two steps. First, with symbolic computations, a Groebner basis or resultants are calculated, and as a second step numerical calculations take place. For testing and prototyping this approach is very useful.

In this article a different situation is considered, that is, we assume that some of the constants in (1) are symbolic and we assume that (1) has to be solved for many thousands of problem instances per second for different values of the symbolic constants. Then, for the sake of speed, one is forced to program the algorithms in a conventional programming language, say C [1]. It is not acceptable, however, to copy the complete process as performed by the computer algebra system into C because the symbolic computations may be very time consuming, even in C. Therefore, the symbolic computations should be performed in a preprocessing phase by the computer algebra system, and the resulting equations should, for every problem instance, be solved by numerical methods implemented in C.

1.1 The conventional approach and its problems in more detail

In more detail, the conventional approach means that from (1) a univariate polynomial, say $p(x)$, is derived by MAPLE. Obviously, $p(x)$ will contain symbolic constants. Now $p(x) = 0$, $f(x, y) = 0$ and $g(x, y) = 0$ are copied into the C program and for every problem instance the constants in these equations are replaced by numerical values. In general, solving $p(x) = 0$ is straightforward, giving the roots $x_1...x_n$. Subsequently, for every root x_i the corresponding root y_i has to be determined. To that end, x_i is backsubstituted in $f(x, y) = 0$ and $g(x, y) = 0$, giving the univariate polynomial equations $f(x_i, y) = 0$ and $g(x_i, y) = 0$. Solving $f(x_i, y) = 0$ for y gives the solutions $y_{f1}...y_{fm}$, and solving $g(x_i, y) = 0$ for y gives the solutions $y_{g1}...y_{gk}$. The value y_i occurring both in $y_{f1}...y_{fm}$ and $y_{g1}...y_{gk}$ is the desired value, i.e., the pair x_i, y_i is a root of (1). During this process, a number of complications may occur:

1. The equation $f(x_i, y) = 0$ may be degenerate, i.e. may be $0 = 0$, or even worse, may be near degenerate within the noise margin. The case of exact degeneracy is easily detected but it is not trivial to detect near degeneracy. In both cases every solution of the other equation, that is, of $g(x_i, y) = 0$ is a correct root. Analogously, $g(x_i, y) = 0$ may be degenerate, giving the same problems. As we know that (1) has a finite number of solutions the situation that $f(x_i, y) = 0$ and $g(x_i, y) = 0$ are both degenerate will not occur.
2. It is sometimes hard to select from $y_{f1}...y_{fm}$ and $y_{g1}...y_{gk}$ the collective value y_i because, by numerical errors, the actual value of y_i will be different in the two sets.
3. $p(x) = 0$ may have multiple roots, that is, the roots $x_1...x_n$ may contain (near) identical values. Let us assume that there is a double root, given by the the identical values x_j and x_{j+1}. Then there will be two matching roots y_j and y_{j+1}, not necessarily with the same value. When y_j is matched to x_j, in a later stage not y_j should be matched to x_{j+1} but y_{j+1}.

The problems mentioned in 1 are the hardest to detect and to handle. When this is not done properly, solutions may be lost or may be completely wrong. None of the problems in 1,2,3 can be foreseen, prevented or solved during the

[1] Of course, instead of MAPLE and C every other computer algebra system and programming language may be used.

preprocessing stage, they have to be dealt with during the numerical calculations. On the other hand, none of these problems is insurmountable. For example, when the problem in 1 occurs, in general the substitution $u = x+y$, $v = x-y$ gives non-degenerate equations. Also, by using exact computations, interval arithmetic or floating point filters the problem in item 2 can be avoided, and by bookkeeping properly the problem in item 3 is solved. However, apart from the speed penalty introduced by these techniques, the complexity of the implementation increases significantly by these measures. Moreover, every new set of equations (1) should be analysed, possible degeneracies should be signalled, and appropriate measures should be devised and implemented to handle these degeneracies. The method we propose avoids these complications. It is simple and robust, and most of all it is universal, i.e. no problem specific programming is required. It only assumes that two univariate polynomials $p(x)$ and $q(y)$ can be derived from (1), and it uses the minimum weight perfect bipartite matching algorithm.

For more than two equations a more general combinatorial optimization technique is required namely the multidimensional assignment method. Recently a new algorithm for this problem has been proposed [1,2] and in the last section it is discussed to which extent it may be used for more than two equations. Because the crucial part of our method is a combinatorial optimization technique we call it the CORS method, which stands for Combinatorial Optimization Root Selection method.

Literature The CORS method is not mentioned in the literature. In [3] a method is discussed that avoids part of the problems in 1,2, and 3, but it only works for exact arithmetic. Essence of that method is that the root y_j, matching the root x_i, is found by substituting x_i in (1), and to calculate of these equations the greatest common devisor (GCD) with for example Euclid's algorithm. The root of the GCD is the desired value y_j.

Packages Some numerical packages in C exists to solve (1) as for example SYNAPS [4]. The disadvantage of these packages is however that the symbolic computations are not handled in a preprocessing phase, which results in a significant decrease of the performance, but most of all, sometimes solutions are missed due to the problems mentioned earlier.

Not in this article This article is not about a numerical method to obtain more precise results. It is about robustness. Solutions that are missed by other methods are found with the CORS method.

Article structure In section 2 the minimum weight perfect bipartite matching is introduced and it is shown how this may be used to select roots. In section 3 the implementation and the results of CORS are discussed. In section 4 it is discussed how the CORS method may be generalized to more than two equations.

1.2 An example problem

To see how (1) is solved using the conventional approach let us look at the following small example.

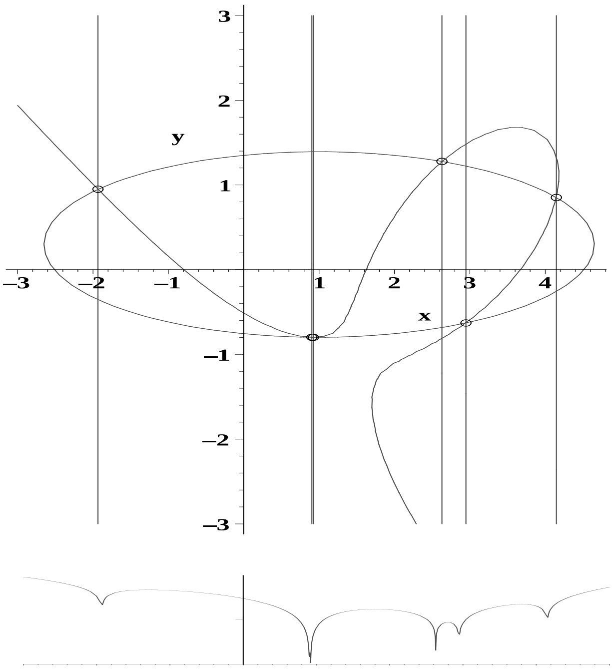

Fig. 1. The conventional method to solve (1). Lower part: the univariate polynomial $p(x)$ plotted logarithmically (in order to limit the dimension in the y direction). The local minima represent the roots of $p(x)$. Upper part: the curves $f(x, y)$ and $g(x, y)$, and vertical lines at positions corresponding with the roots of $p(x)$. To find the intersections of the curves, for every vertical line the intersections with the curves are calculated and identical y-values are selected. For more details see the main text.

$$f(x, y) = (x - a)^3 + (y + b)^3 - 5\ (x - a)\ (y + b) - c = 0$$
$$g(x, y) = (0.3\, x - d)^2 + (y - e)^2 - h = 0 \tag{2}$$

with $a = 1.5$, $b = 1$, $c = 0.4$, $d = 0.3$, $e = 0.29756$ and $h = 1.2$. For these constants the system has four single real roots and a double real root, see figure 1.

As an example, let us determine the root y_i, matching the root $x = 4.1481$ which is the rightmost x-root in this figure. The line $x = 4.1481$ intersects $f(x, y)$ three times, giving $y_{f1} = -5.192317423$, $y_{f2} = 0.8525335317$ and $y_{f3} = 1.339783891$ The line $x = 4.1481$ intersects $g(x, y)$ twice, giving $y_{g1} = 0.8525335293$ and $y_{g2} = -0.2574135293$. Hence, $y = 0.8525335$ is the root matching $x = 4.1481$. Clearly, $y_{f2} = 0.8525335317$ does not equal $y_{g1} = 0.8525335293$, which demonstrates point 3.

The CORS method will be tested on these equations and on the equations that were the starting point of this research, that is, the equations resulting from a computational geometry problem [5], given by

$$\begin{aligned} a_1u^2w^2 + b_1u^2w + c_1uw^2 + d_1u^2 + e_1w^2 + f_1u + g_1w + h_1 = 0 \\ a_2u^2w^2 + b_2u^2w + c_2uw^2 + d_2u^2 + e_2w^2 + f_2u + g_2w + h_2 = 0. \end{aligned} \quad (3)$$

Here, $a_1..h_1$ and $a_2..h_2$ are real constants, as they occur in the computational geometry problem.

2 Selecting roots by constructing a minimal-weight matching

2.1 The bipartite weighted graph and matching

A complete weighted bipartite graph $G = (V, E, w)$ consists of a set of vertices $V = V_1 \cup V_2$, $|V_1| = |V_2| = n$, and of a set of n^2 arcs $(i, j) \in E \subseteq V_1 \times V_2$ with weights $w(i, j)$ for all $(i, j) \in E$. We define a feasible matching π as a permutation which maps V_1 onto V_2 and the weight of permutation π is $w(\pi) = \sum_{(i,j)\in\pi} w(i, j)$. The Minimum Weighted Bipartite Matching Problem is the problem of finding $\pi_0 \in \arg\min\{w(\pi) : \pi \in \mathbb{P}\}$. Here $\mathbb{P}$ is the set of all permutations, and π is called *feasible* if $w(\pi) < \infty$. This problem is well-known in the field of combinatorial optimization and is called the *Linear Assignment Problem* (LAP). Many efficient algorithms exist for solving the LAP with time complexity $O(n^3)$, see [8] and references within. All these algorithms are based on the shortest augmenting path method, the implementation of which, for example in the Hungarian algorithm, is based on the Koning-Egervary theorem [8].

In figure 2 part of a complete bipartite weighted graph is shown with four vertices in V_1 and four in V_2, and two matchings derived from this graph are shown. The rightmost matching has minimum weight.

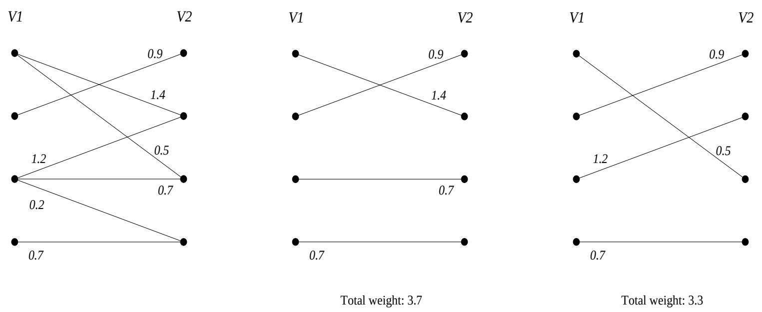

Fig. 2. Left: a weighted bipartite graph G with four vertices in V1 and in V2, and with seven arcs. Middle: a possible matching of G. Right: the minimum weight matching of G.

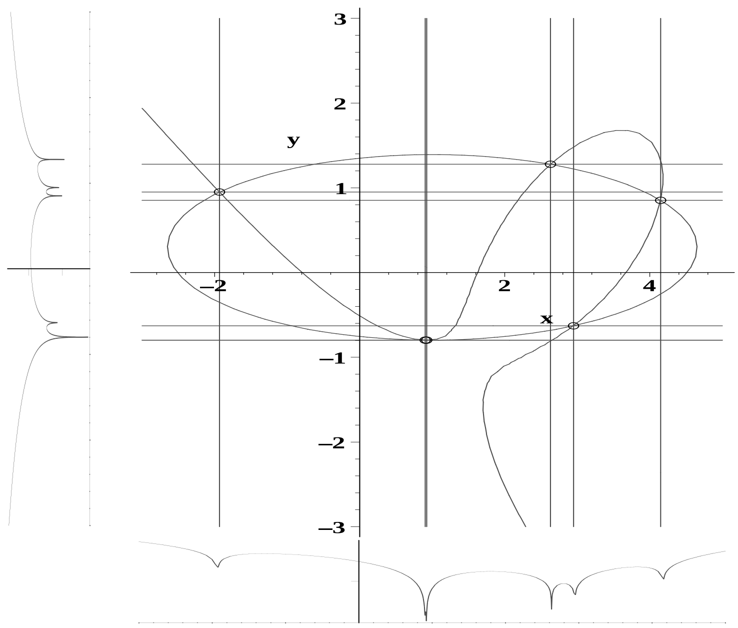

Fig. 3. The CORS method explained graphically. The figure consists of three parts. Lower part: the univariate polynomial $p(x)$ plotted logarithmically. The five local minima represent the roots of $p(x)$. The root at $x = 0.912$ is a double root. Left part: the univariate polynomial $q(y)$ plotted logarithmically. The five local minima represent the roots of $q(y)$. The root at $y = -0.797$ is a double root. Main part: the curves $f(x, y)$ and $g(x, y)$. The vertical lines are at positions corresponding with the roots of $p(x)$ and the horizontal lines are at positions corresponding with the roots of $q(y)$. The intersections of the curves are found by intersecting the horizontal lines with the vertical lines and selecting from all 36 intersections six intersections. The selection process is performed by calculating the minimum weight perfect bipartite matching. For more details see the main text.

2.2 Selecting the roots by using a bipartite matching

Let us now see how a bipartite weighted graph and a minimum weight matching of this graph is used to select the roots of two polynomial equation. We consider again the polynomials with symbolic constants in (1). A computer algebra system is used to eliminate from (1) y and x, giving two univariate polynomials $p(x)$ and $q(y)$ respectively. Whether this done by calculating two Groebner basis or by calculating two resultants is irrelevant. Now $p(x)$, $q(y)$, $f(x, y)$ and $g(x, y)$ are copied into the C code. In these four equations, for every problem instance, the symbolic constants are replaced by numerical values and the roots in $\mathbb{C}$ of $p(x)$ and $q(y)$ are calculated numerically. Both $p(x)$ and $q(y)$ have n roots represented by $x_1..x_n$ and $y_1..y_n$ respectively. These roots are used to calculate n^2 weights, where $w_{i,j}$ is defined as

$$w_{i,j} = \sqrt{(|(f(x_i, y_j))^2| + |(g(x_i, y_j))^2|)}. \tag{4}$$

Subsequently a complete weighted bipartite graph G is constructed. The nodes in V_1 consist of the values $x_1..x_n$, and the nodes in V_2 consist of the values $y_1..y_n$. The arc between node x_i and y_j is assigned the weight $w_{i,j}$. Now of G the minimum-weight perfect matching π_0 is calculated. The n arcs in π_0 represent the optimal solutions of (1). Here, optimal means that the sum of the errors is minimal. In the following this method of root selection is called CORS1.

Instead of minimizing the sum of the errors it is also possible to minimize the maximum error. This is done by adjusting the weights in G as follows. All n^2 arcs and their weights are stored in a linear list L. Subsequently, L is sorted in increasing order of the weight. Now the weight in the first entry in L is set to 1, and the weight of item i is set to the sum of the weights in previous items plus one, i.e. $weight[i] = (\sum_{j=1}^{i-1} weight[j]) + 1$. Table (5) is generated with this method. In this way the matching is forced to consist of a subset of the first M items in L, with M as small as possible. We call this method CORS2.

3 Implementation, tests and results

Implementation From (3) the univariate polynomials $p(u)$ and $q(w)$ are derived with MAPLE. The numerical calculations are implemented in C++ in double precission. Laguerre's method [6] is used to compute the roots of the polynomials $p(u)$ and $q(w)$. The LEDA [7] implementation of the minimum weight bipartite matching algorithm is used. Problem instances are obtained by randomly generating instances of the computational geometry problem [5], giving the constants $a_1..h_1$ and $a_2..h_2$.

Tests We tested the CORS1 and CORS2 method on (2) and on (3). Only the results of the latter problem are presented here, the results of (2) are similar. Every problem instance is solved in two ways: with the CORS method and with SYNAPS, a C++ package for solving polynomial equations. We solved 10^5 problem instances with CORS and SYNAPS, and ≈ 400 with MAPLE. The latter problem instances were solved correctly by CORS and were missed by SYNAPS, i.e. we use MAPLE to decide whether CORS or SYNAPS gave the correct result. **Results** In general the results of CORS1 and CORS2 are identical. In the tests approximately 2.4% of the solutions is missed by SYNAPS and are found by CORS. This is confirmed by solving these problem instances with MAPLE. No solutions were missed by CORS. The average error of the solutions found by SYNAPS is 1.3 10^{-10} and of CORS 6.5 10^{-11}. Running 10^5 problem instances with CORS takes 14 *sec.* and with SYNAPS 475 *sec.*

4 Applying the CORS method to more than two equations

The CORS method can easily be implemented for a system of two polynomial equations because many efficient implementations of the minimum weight bipartite matching algorithm exist. However, multidimensional matching algorithms are less abundant. Therefore, we now propose the Tolerance Based Algorithm (TBA) for solving the d-Dimensional Matching Problem (DMP). The idea of this algorithm is similar to the idea of a recently designed algorithm for solving the ATSP [2]. For sake of simplicity we first outline the TBA for the 2-DMP.

The algorithm is based on the tolerances for the Relaxed LAP (RLAP). A *feasible solution* θ to the RLAP is a mapping θ of V_1 into V_2 with $w(\theta) < \infty$. A feasible solution π to the LAP can be represented by a set of n arcs (i, j) such that the out-degree $d(i) = 1$ for all $i \in V_1$ and the in-degree $d(j) = 1$ for all $j \in V_2$, and a feasible solution θ to the RLAP can be represented by a set of n arcs (i, j) with the out-degree $d(i) = 1$ for all $i \in V_1$ and $\sum_{\ j \in V_2} d(j) = n$. We denote the set of all feasible solutions to the RLAP by $\mathbb{Q}$. It is clear that $\mathbb{P} \subset \mathbb{Q}$. Note that θ is feasible to the LAP if the in-degree $d(j) = 1$ for all $j \in V_2$. The RLAP is the problem of finding $\min\{w(\theta) : \theta \in \mathbb{Q}\} = w(\theta_0) \leq w(\pi_0)$ which is a lower bound $w(\theta_0)$ of $w(\pi_0)$.

The *tolerance problem* for the RLAP is the problem of finding , for each $e \in E$, the maximum decrease $l(e)$ and the maximum increase $u(e)$ of the arc weight $w(e)$ preserving the optimality of θ_0 under the assumption that the weights of all other arcs remain unchanged. The values $l(e)$ and $u(e)$ are called the *upper* and the *lower tolerances,* respectively, of edge e with respect to the optimal solution θ_0 and the function of arc weights w. For a selected arc, i.e an arc $[i_1, j_1(i_1)] \in \theta_0$ we define the *upper tolerance* $u[i, j_1(i)] = w[i, j_2(i)] - w[i, j_1(i)]$ with $w[i, j_1(i)] \leq w[i, j_2(i)] \leq ... \leq w[i, j_n(i)]$, and the *lower tolerance* $l[i, j_1(i)] = \infty$. Similarly, for an unselected arc, i.e an arc $[i^1(j), j] \notin \theta_0$ the *lower tolerance* $l[i^1(j), j] = w[i^1(j), j] - w[i^1(j), j_1(i^1(j))]$ with $w[i^1(j), j] \leq w[i^2(j), j] \leq ... \leq w[i^n(j), j]$, and the *upper tolerance* $l(i, j) = \infty$. The *bottleneck tolerance value* $b(\theta) = \max\{u(\theta), l(\theta)\}$ is defined as follows. For each $j \in V_2$, $u(j) = \min\{u(i, j) :$ for all (i, j) with $\deg(j) > 1\}$, and $u(\theta) = \max\{u(j)$:for all j with $\deg(j) > 1\}$. Similarly, for each $j \in V_2$, $l(j) = \min\{l(i, j)$:for all (i, j) with $\deg(j) = 0\}$, $l(\theta) = \max\{l(j)$:for all j with $\deg(j) = 0$. Further we treat each π, and each θ as the sets of corresponding arcs such that $|\pi| = |\theta| = n$.

The TBA is based on the following results [1,2]:
(i) if $\theta_0 \notin \mathbb{P}$, then $w(\theta_0) + b(\theta_0) \leq w(\pi_0)$.
(ii) if $\mathbb{Q}_0 = \{\theta : w(\theta) = w(\theta_0)\}$ and $u(i,j) > 0$ for all $(i,j) \in \theta_0$ then $|\mathbb{Q}_0| = 1$.
(iii) if $u(i,j) > 0$ for $(i,j) \in \alpha \subset \theta_0$ and $u(i,j) = 0$ for $(i,j) \in \theta_0 \backslash \alpha$, then $|\mathbb{Q}_0| > 1$ and $|\cap \mathbb{Q}_0| = |\alpha|$. and can be outlined as follows.

Based on these properties of the RLAP, the LAP may be solved using the following tolerance based algorithm

1. Find θ_0.
2. If $\theta_0 \in \mathbb{P}$, then $\pi_0 = \theta_0$, and output π_0 with $w(\pi_0)$, stop.
3. Find $b(\theta)$.
4. If $b(\theta) = u[i, j_1(i)]$, then replace in θ_0 the arc $[i, j_1(i)]$ by the arc $[i, j_2(i)]$; otherwise (if $b(\theta) = l[i^1(j), j]$) replace in θ_0 the arc $[i^1(j), j_1(i^1(j))]$ by the arc $[i^1(j), j]$.
5. Return to step 2.

Note that for CORS2, after each iteration of the TBA for solving the LAP, the current number of vertices $j \in V_2$ with $d(j) = 1$ will be increased by at least one vertex. Hence, the time complexity of the TBA is $O(n^3)$.

In the following example we illustrate the TBA for the complete weighted bipartite graph below

8	16384	256	4096
2	16	4	64
512	32768	128	32
1	1024	8192	2048

(5)

1. $\theta_0 = \{(1,1),(2,1),(3,4),(4,1)\}$; 2. $\theta_0 \notin \mathbb{P}$; 3. $b(\theta) = 14$; 4. $b(\theta) = l(2,2)$, replace in in θ_0 the arc $(2,1)$ by $(2,2)$. 5. Return to step 2. 2. $\theta_0 = \{(1,1),(2,2),(3,4),(4,1) \notin \mathbb{P}$; 3. $b(\theta) = 248$; 4. $b(\theta) = u(1,1)$, replace in in θ_0 the arc $(1,1)$ by $(1,3)$; 5. Return to step 2. 2. $\theta_0 = \{(1,3),(2,2),(3,4),(4,1) \in \mathbb{P}$, then $\pi_0 = \theta_0$, and output π_0 with $w(\pi_0) = 305$, stop.

This tolerance based algorithm may be generalized to more than two dimensions as follows. In case of d-DMP $V = \cup_{i=1}^{d} V_i$, $|V_i| = n$, the set of arcs $e \in E \subseteq \otimes_{i=1}^{d} V_i$ with weights $w(e)$ for all $e \in E$, and the in-degree $d(j)$ for all arcs (i,j) will be defined similarly to the 2-DMP with replacing V_2 by $\overline{V_2}$ which is a projection of the $\otimes_{i=2}^{d} V_i$ on V_2. If the objective function of d-DMP is to minimize the maximim error of all n matched roots, then TBA has $O((d-1)n^3)$ time complexity.

Currently we are implementing the multidimensional tolerance based matching algorithm, and we will test it on a system of three polynomial equations. For the 3D case we can not use Sylvester resultants to calculate the univariate polynomials $p(x)$, $q(y)$ and $r(z)$, we have to use a multiresultant method. These are widely available. No other algorithms are required to test the CORS method on three polynomial equations.

5 Discussion and conclusion

As mentioned before, the CORS method is not meant to refine numerical results but to find solutions that are missed by other methods. The numerical quality of the solutions found by CORS depends on the quality of the univariate polynomial solver. We used Laguerre's method, but for multiple roots of even degree it is probably better to use Broydens method [6]. In spite of using a non-optimal univariate polynomial solver, our experiments show that the CORS method outperforms current methods in the sense that it does not lose roots. Moreover, it is faster and more acurate.

The CORS method is only suitable for systems of d equations with N solutions with small d and small N. That is because the number of arcs in the graph G is N^d. We think that the CORS method is efficient for $d \leq 3$ and $N \leq 20$, and may be used to analyze a system of equations with $d \leq 5$ and $N \leq 30$. Probably, for larger systems the overhead will be unacceptable.

Many problems in engineering and technology boil down to solving a small system of polynomial equations. Using the CORS method, only simple univariate equations have to be solved. Selecting the correct roots with a combinatorial optimization method proves to work well. In this way the user does not have to worry about degeneracies and other complications.

Literature

[1] Goldengorin, B., Sierksma, G. *Combinatorial optimization tolerances calculated in linear time.* SOM Research Report 03A30, University of Groningen, Groningen, The Netherlands, 2003(http://www.ub.rug.nl/eldoc/som/a/03A30/03a30.pdf).

[2] Goldengorin, B., Sierksma, G., and Turkensteen, M. *Tolerance Based Algorithms for the ATSP. Graph-Theoretic Concepts in Computer Science.* 30th International Workshop, WG 2004, Bad Honnef, Germany, June 21-23, 2004. Hromkovic J., Nagl M., Westfechtel B. (Eds.). Lecture Notes in Computer Science Vol. 3353, pp. 222-234, 2004. .

[3] Sederberg, T., W., Zheng, J. *Algebraic methods for computer aided geometric design* in *Handbook of computer aided geometric design* Farin, G., Hoschek, J., Kim, M., S., (Eds.), North-Holland Elsevier (2002) p. 378.

[4] Synaps. Available at: http://www.inria.fr/galaad/logiciels/synaps/inex.html

[5] Bekker, H., Roerdink, J. B. T. M. *Calculating critical rotations of polyhedra for similarity measure evaluation.* Proceedings of IASTED International conference on Computer Graphics and Imaging, Palm springs, October 1999.

[6] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. *Numerical Recipes in C++, the Art of Scientific Computing.* Cambr. Univ. Press, New York.

[7] K. Melhorn, Näher, S. *LEDA A Platform for Combinatorial and Geometric Computing.* Cambridge University press,Cambridge. 1999

[8]Burkard, R.E. *Slected topics on assignment problems.* Discrete Applied Mathematics 123 (2002) 257–302.