Solving Polynomial Systems using PHCpack

Jan Verschelde

Department of Math, Stat & CS
University of Illinois at Chicago
Chicago, IL 60607-7045, USA

email: jan@math.uic.edu

URL: http://www.math.uic.edu/~jan

Geometry of Mechanism Science (GeMS'07)

University of Notre Dame, 1 - 3 March 2007

Mission

- solve polynomial systems arising in applications
  → turn applications into benchmarks
- develop convenient interfaces to PHCpack
  → blackbox, toolbox, PHCmaple, PHClab, C
- run on multiprocessor computers and clusters
  → solve systems with $>$ 100,000 solutions

Polynomial Systems in Applications

What are we solving?

- polynomial systems: format of our input
  the study of its solutions = algebraic geometry
- applications: relevance to science & engineering
  our application field concerns mechanical design
- benchmarks: test performance of our methods
  one goal is to turn applications into benchmarks

About PHCpack

PHC = Polynomial Homotopy Continuation

- Version 1.0 archived as Algorithm 795 by ACM TOMS.
- Platform for numerical algebraic geometry.
- Pleasingly parallel implementations
  + **Yusong Wang** of Pieri homotopies (HPSEC'04);
  + **Anton Leykin** of monodromy factorization (HPSEC'05);
  + **Yan Zhuang** of polyhedral homotopies (HPSEC'06).
- Some current developments:
  + **Yun Guan:** PHClab, experiments with MPITB in Octave;
  + **Kathy Piret:** bindings with Python for use in SAGE.

Some Benchmarks from Mechanical Design

name	n	D	B_Z	B_S	V	#sols
assur44	8	864	128	128	128	56
fbremb2	6	256	240	163	96	68
fbrfive12	12	4096	96	96	36	36
fbrfive4	4	256	96	194	36	36
fourbar	4	256	96	96	80	36
ipp	8	256	96	96	64	48
ipp2	11	1024	576	848	288	16
kin1	12	4608	320	896	192	48
kinema	9	64	240	64	64	40
puma	8	128	16	32	16	16
rbp1	6	486	160	160	160	40
rbpl24	9	576	80	80	80	40
stewgou40	9	4,096	2,560	2,560	1,536	40
rps10	10	262,144	18,432	9,216	1,024	1,024

different formulations of the same problem matter for solving

most problems are now of little use as challenging benchmarks

Five-Point Path Synthesis

Design a 4-bar linkage = design trajectory of coupler point.

Input:	coordinates of points on coupler curve.
Output:	lengths of the bars of the linkage.

C.W. Wampler: **Isotropic coordinates, circularity and Bezout numbers: planar kinematics from a new perspective.**
Proceedings of the 1996 ASME Design Engineering Technical Conference.
Irvine, CA, Aug 18–22, 1996.

A.J. Sommese and C.W. Wampler: **The Numerical Solution of Systems of Polynomials Arising in Engineering and Science.**
World Scientific, 2005.

Isotropic Coordinates

- A point $(a, b) \in \mathbb{R}^2$ is mapped to $z = a + ib$, $i = \sqrt{-1}$.
- $(z, \bar{z}) = (a + ib, a - ib) \in \mathbb{C}^2$ are *isotropic coordinates*.
- Observe $z \cdot \bar{z} = a^2 + b^2$.
- Rotation around $(0, 0)$ through angle θ is multiplication by $e^{i\theta}$. Multiply by $e^{-i\theta}$ to invert the rotation.
- Abbreviate a rotation by $\Theta = e^{i\theta}$, then its inverse $\Theta^{-1} = \bar{\Theta}$, satisfying $\Theta\bar{\Theta} = 1$.

The Loop Equations

Let $A = (a, \bar{a})$ and $B = (b, \bar{b})$ be the fixed base points.

Unknown are $(x, \bar{x})$ and $(y, \bar{y})$, coordinates of the other two points in the 4-bar linkage.

For given precision points $(p_j, \bar{p}_j)$, assuming $\theta_0 = 1$,

$$\begin{cases} (p_j + x\theta_j + a)(\bar{p}_j + \bar{x}\bar{\theta}_j + \bar{a}) = (p_0 + x + a)(\bar{p}_0 + \bar{x} + \bar{a}) \\ (p_j + y\theta_j + b)(\bar{p}_j + \bar{y}\bar{\theta}_j + \bar{b}) = (p_0 + y + b)(\bar{p}_0 + \bar{y} + \bar{b}) \end{cases}$$

Since the angle θ_j corresponding to each $(p_j, \bar{p}_j)$ is unknown, five precision points are needed to determine the linkage uniquely.

Adding $\theta_j \bar{\theta}_j = 1$ to the system leads to 12 equations in 12 unknowns: $(x, \bar{x})$, $(y, \bar{y})$, and $(\theta_j, \bar{\theta}_j)$, for $j = 1, 2, 3, 4$.

```
for k from 1 to 4 do
  eq[k] := theta[k]*Theta[k] - 1;
  eq[k+4] := (p[k] + x*theta[k] + a)*(P[k] + X*Theta[k] + A)
         - (p[0] + x + a)*(P[0] + X + A);
  eq[k+8] := (p[k] + y*theta[k] + b)*(P[k] + Y*Theta[k] + B)
         - (p[0] + y + b)*(P[0] + Y + B);
end do;
sys := [seq(eq[k],k=1..12)]; s := map(expand,sys);
for k from 1 to 4 do
  ss[k] := s[k];
  ss[k+4] := expand(algsubs(theta[k]*Theta[k]=1,s[k+4]));
  ss[k+8] := expand(algsubs(theta[k]*Theta[k]=1,s[k+8]));
end do;
points := seq([stats[random,uniform[-1,+1]](2)],k=1..7);
for k from 1 to 5 do
  p[k-1] := points[k][1] + points[k][2]*I; P[k-1] := points[k][1] - points[k][2
end do;
a := points[6][1] + points[6][2]*I; A := points[6][1] - points[6][2]*I;
b := points[7][1] + points[7][2]*I; B := points[7][1] - points[7][2]*I;
writeto("/tmp/fbrfive");
for k from 1 to 12 do lprint(expand(ss[k])); printf(";\n"); end do;
```

```
12
theta[1]*Theta[1]-1;
theta[2]*Theta[2]-1;
theta[3]*Theta[3]-1;
theta[4]*Theta[4]-1;
-.4091256991*x*theta[1]-1.061607555*I*x*theta[1]+1.157260179-.3374636810*X+.1524877812*I*X
   -.3374636810*x-.1524877812*I*x-.4091256991*X*Theta[1]+1.061607555*I*X*Theta[1];
.4011300738*x*theta[2]-1.146477955*I*x*theta[2]+1.338182778-.3374636810*X+.1524877812*I*X
   -.3374636810*x-.1524877812*I*x+.4011300738*X*Theta[2]+1.146477955*I*X*Theta[2];
.3705985316*x*theta[3]-1.454067014*I*x*theta[3]+2.114519894-.3374636810*X+.1524877812*I*X
   -.3374636810*x-.1524877812*I*x+.3705985316*X*Theta[3]+1.454067014*I*X*Theta[3];
.3188425748*x*theta[4]-.850446965*I*x*theta[4]+.6877863684-.3374636810*X+.1524877812*I*X
   -.3374636810*x-.1524877812*I*x+.3188425748*X*Theta[4]+.850446965*I*X*Theta[4];
-1.742137552*y*theta[1]-.3932004150*I*y*theta[1]+1.524665181+.9955481716*Y+.8208949212*I*Y
   +.9955481716*y-.8208949212*I*y-1.742137552*Y*Theta[1]+.3932004150*I*Y*Theta[1];
-.9318817788*y*theta[2]-.4780708150*I*y*theta[2]-.5680292799+.9955481716*Y+.8208949212*I*Y
   +.9955481716*y-.8208949212*I*y-.9318817788*Y*Theta[2]+.4780708150*I*Y*Theta[2];
-.9624133210*y*theta[3]-.7856598740*I*y*theta[3]-.1214837957+.9955481716*Y+.8208949212*I*Y
   +.9955481716*y-.8208949212*I*y-.9624133210*Y*Theta[3]+.7856598740*I*Y*Theta[3];
-1.014169278*y*theta[4]-.1820398250*I*y*theta[4]-.6033068118+.9955481716*Y+.8208949212*I*Y
   +.9955481716*y-.8208949212*I*y-1.014169278*Y*Theta[4]+.1820398250*I*Y*Theta[4];
```

Output of phc -b on 5-Point Synthesis Problem

```
total degree : 4096
6-homogeneous Bezout number : 96
general linear-product Bezout number : 96
mixed volume : 36
solution 36 :    start residual :  1.672E-15   #iterations : 1   success
t :  1.00000000000000E+00   0.00000000000000E+00
m : 1
the solution for t :
 theta[1] :  3.04923062675137E+00  -1.36666126486689E+01
 Theta[1] :  1.55514190369546E-02   6.97012611150467E-02
 theta[2] :  1.94158500874355E-01  -1.70861689159530E+00
...
 Y :  7.72626833143914E-01  -4.06259823401552E-01
== err :  7.607E-14 = rco :  3.915E-04 = res :  1.439E-15 = complex regu
```

Summary of Output of phc -b

```
== err :  7.607E-14 = rco :  3.915E-04 = res :  1.439E-15 = complex regu
============================================================
Frequency tables for correction, residual, condition, and distances :
FreqCorr :  0 0 0 0 0 0 0 0 0 0 0 1 8 17 0 10 : 36
FreqResi :  0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 33 : 36
FreqCond :  0 10 15 9 0 2 0 0 0 0 0 0 0 0 0 0 : 36
FreqDist :  36 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 : 36
Small correction terms and residuals counted to the right.
Well conditioned and distinct roots counted to the left.

  ---------------------------------------------------------------
  |   root counts  |  start system  |  continuation  |   total time   |
  ---------------------------------------------------------------
  |  0h 0m 4s910ms |  0h 0m 7s570ms |  0h 0m 8s 60ms |  0h 0m20s720ms |
  ---------------------------------------------------------------
```

PHClab: A MATLAB/Octave Interface

with Yun Guan

- easier to use `phc` via automatic conversions
  of formats for polynomial systems and solutions
- adds scripting capabilities to PHCpack in a standard way
- solving many polynomial systems in parallel
  via the MPI toolbox for Octave:

→ this use of `phc` provides automatic quality control

Design of PHClab

- User only needs executable version of `phc`.
- PHClab function `set_path`(*location of phc*) is executed at the beginning of a MATLAB or Octave session.
- The scripts that implement the functions of PHClab call `phc` with the appropriate options and with redirected input and output from and to temporary files.

PHClab functions for isolated solutions

`solve_system`	calls the blackbox solver
`refine_sols`	refines the solutions by application of Newton's method
`mixed_volume`	computes the mixed volumes and solve a random system
`track`	given target and start system + solutions, tracks paths
`deflation`	applies deflation for isolated singularities

Note: `mixed_volume` calls translated code from

T. Gao, T.Y. Li, and M. Wu. Algorithm 846:
MixedVol: a software package for mixed-volume computation.
ACM Trans. Math. Softw., 31(4):555–560, 2005.

numerical irreducible decomposition in PHClab

- top down:

`embed`	adds extra hyperplanes and slack variables
`cascade`	performs a cascade of homotopies

- filtering and factoring:

`phc_filter`	filters junk on higher dimensional components
`decompose`	partitions a witness set along irreducible factors

- bottom up:

`eqnbyeqn`	an equation-by-equation solver
`intersection`	intersection of two witness sets

scripts on Griffis-Duffy system

top down:

```
S = read_system('gdplatB');    % read the system from file
E = embed(S,1);                % embed with one extra hyperplane
sols = solve_system(E);        % call the blackbox solver
size(sols,2)                   % to see candidate witness #points
[sw,R] = cascade(E,sols)       % perform a cascade
```

bottom up:

```
p = read_system('gdplatBa')    % read the system from file
[sw,R] = eqnbyeqn(p)           % call equation-by-equation solver
```

process witness sets:

```
dc = decompose(R{2},sw{2,1})   % decompose into irreducible factor
```

Using MPI Toolbox in Octave

- suppose one needs to solve hundreds of polynomial systems and a cluster computer is available
- requires dynamically linked LAM/MPI libraries and a custom compiled Octave
- use dynamic load balancing to distribute the systems among the worker nodes in the cluster
- used to solve 183 "origami equation systems" posed at problem session at IMA software workshop last fall
- with script in `origamipar_v2`, we run it typing

```
mpirun -c 14 octave-2.1.64 -q --funcall origamipar_v2
```

J. Fernández, M. Anguita, E. Ros, and J.L. Bernier. SCE Toolboxes for the development of high-level parallel applications. LNCS 3992, pages 518–528, Springer 2006.

ways to compute in parallel

1. batch jobs

   submit a job and wait for it to return

2. interactive

   small number of processors on large supercomputer
   or on a cluster, asking for user input, interaction

3. via scripting

   use MPI toolbox in Octave

Difficulties for Parallel PHCpack

- Optimal reuse of the existing software.
- Parallel code is just scheduling of jobs.
- Avoid storing all start solutions in main memory.
- Deal with large output sets.

An ambitious Swap of Letters:

PHC = Polynomial Homotopy Continuation

HPC = High Performance Computing

towards High Performance *Continuation*

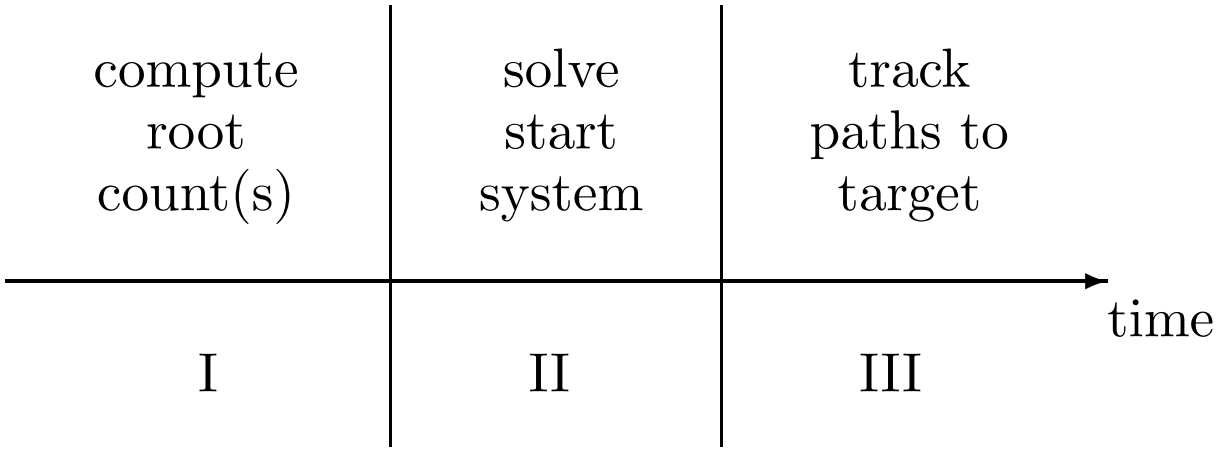
Other Parallel Homotopy Solvers

T. Gunji, S. Kim, K. Fujisawa, and M. Kojima:
PHoMpara – parallel implementation of the Polyhedral Homotopy continuation Method for polynomial systems. *Computing* 77(4):387–411, 2006.

H.-J. Su, J.M. McCarthy, M. Sosonkina, and L.T. Watson:
Algorithm 857: **POLSYS_GLP**: A parallel general linear product homotopy code for solving polynomial systems of equations. *ACM Trans. Math. Softw.* 32(4):561–579, 2006.

design of phc as a toolbox

Roughly, there are three stages when solving a polynomial system using polynomial homotopy continuation:



Usually stage III is most time consuming.

But if millions of start solutions, memory gets too full...

Applying Program Inversion to Homotopy Solver

$$h(\mathbf{x},t) = \gamma(1-t)g(\mathbf{x}) + tf(\mathbf{x}) = \mathbf{0}, \quad \gamma \in \mathbb{C}, \quad t \in [0,1].$$

Input: $g(\mathbf{x}) = \mathbf{0}$;
for k from 1 to $\#g^{-1}(\mathbf{0})$ do
    compute $\mathbf{y}_k$: $g(\mathbf{y}_k) = \mathbf{0}$;
end for;
output: $g^{-1}(\mathbf{0})$.

solve start system

track paths to target

Input: $g^{-1}(\mathbf{0})$, $h(\mathbf{x},t) = \mathbf{0}$;
for k from 1 to $\#g^{-1}(\mathbf{0})$ do
    path starts at $\mathbf{y}_k \in g^{-1}(\mathbf{0})$;
end for;
output: $f^{-1}(\mathbf{0})$.

Applying Program Inversion to Homotopy Solver

$$h(\mathbf{x},t) = \gamma(1-t)g(\mathbf{x}) + tf(\mathbf{x}) = \mathbf{0}, \quad \gamma \in \mathbb{C}, \quad t \in [0,1].$$

Input: $h(\mathbf{x}) = \mathbf{0}$;
for k from 1 to $\#g^{-1}(\mathbf{0})$ do
    path starts at $\mathbf{y}_k \in g^{-1}(\mathbf{0})$;
end for;
output: $f^{-1}(\mathbf{0})$.

track paths to target

get next start solution

Input: $g(\mathbf{x}) = \mathbf{0}$, k;
compute $\mathbf{y}_k$: $g(\mathbf{y}_k) = \mathbf{0}$;
  or read $\mathbf{y}_k$ from file;
  or type in values for $\mathbf{y}_k$;
output: $\mathbf{y}_k \in g^{-1}(\mathbf{0})$.

Jumpstarting Homotopies with phc

```
[phcpack@idefix bin]$ phc -q
Welcome to PHC (Polynomial Homotopy Continuation) V2.3.24 18 Feb 2007
Tracking Solution Paths with incremental read/write of solutions.

MENU for type of start system or homotopy :
  1. start system is based on total degree;
  2. a linear-product start system will be given;
  3. start system and start solutions are provided;
  4. polyhedral continuation on a generic system;
  5. diagonal homotopy to intersect algebraic sets;
  6. descend one level down in a cascade of homotopies.
Type 1, 2, 3, 4, 5, or 6 to select type :
```

Use PHCpack as a C Library

- Most of the code in PHCpack is in Ada, compiles with gcc.
- The parallel path trackers follow manager-worker protocol.
- The main parallel program is written in C, using MPI.
  Also all routines which handle job scheduling are written in C.
- The C interface uses PHCpack as a state machine:
  1. Feed data into machine and select methods;
  2. Compute with given data and selected methods;
  3. Extract the results from the machine.

  The C user is unaware of the data structures and algorithms.

Jumpstarting Parallel Homotopies

```
[phcpack@idefix bin]$ mpirun -np 14 mpi2track

MENU for type of start system or homotopy :
  1. start system is based on total degree;
  2. a linear-product start system will be given;
  3. start system and start solutions are provided;
  4. the homotopy is a cascade to go one level down;
  5. start extrinsic diagonal homotopy to intersect 2 sets.
Type 1, 2, 3, 4, or 5 to select type of start system :
```

Jumpstarting Homotopies

Problem: huge #paths (e.g.: $> 100{,}000$),

undesirable to store all start solutions in main memory.

Solution: (assume manager/worker protocol)

1. The manager reads start solution from file "**just in time**" whenever a worker needs another path tracking job.
2. For total degree and linear-product start systems, it is **simple to compute** the solutions whenever needed.
3. As soon as worker reports the end of a solution path back to the manager, the solution is **written to file**.

Indexing Start Solutions

The start system $\begin{cases} x_1^4 - 1 = 0 \\ x_2^5 - 1 = 0 \\ x_3^3 - 1 = 0 \end{cases}$ has $4 \times 5 \times 3 = 60$ solutions.

Get 25th solution via decomposition: $24 = 1(5 \times 3) + 3(3) + 0$.

Verify via lexicographic enumeration:

$000 \to 001 \to 002 \to 010 \to 011 \to 012 \to 020 \to 021 \to 022 \to 030 \to 031 \to 032 \to 040 \to 041 \to 042$

$100 \to 101 \to 102 \to 110 \to 111 \to 112 \to 120 \to 121 \to 122 \to \boxed{130} \to 131 \to 132 \to 140 \to 141 \to 142$

$200 \to 201 \to 202 \to 210 \to 211 \to 212 \to 220 \to 221 \to 222 \to 230 \to 231 \to 232 \to 240 \to 241 \to 242$

$300 \to 301 \to 302 \to 310 \to 311 \to 312 \to 320 \to 321 \to 322 \to 330 \to 331 \to 332 \to 340 \to 341 \to 342$

a problem from electromagnetics

posed by Shigetoshi Katsura to PoSSo in 1994:

a family of $n-1$ **quadrics** and one linear equation;
#solutions is 2^{n-1} (= Bézout bound).

$n = 21$**: 32 hours and 44 minutes** to track 2^{20} paths by 13 workers at 2.4Ghz, producing output file of 1.3Gb.

tracking about 546 paths/minute.

verification of output:

1. parsing 1.3Gb file into memory takes 400Mb and 4 minutes;
2. data compression to quadtree of 58Mb takes 7 seconds.

Using Linear-Product Start Systems Efficiently

- Store start systems in their linear-product product form, e.g.:

$$g(\mathbf{x}) = \begin{cases} (x_1 + c_{11}) \times (x_2 + c_{12}x_3 + c_{13}) \times (x_2 + c_{14}x_3 + c_{15}) = 0 \\ (x_2 + c_{21}) \times (x_1 + c_{22}x_3 + c_{23}) \times (x_1 + c_{24}x_3 + c_{25}) = 0 \\ (x_3 + c_{31}) \times (x_1 + c_{32}x_2 + c_{33}) \times (x_1 + c_{34}x_2 + c_{35}) = 0 \end{cases}$$

- Lexicographic enumeration of start solutions,

  $\rightarrow$ as many candidates as the total degree.

- Store results of incremental LU factorization.

  $\rightarrow$ prune in the tree of combinations.

Nine-Point Path Synthesis

H. Alt: Über die Erzeugung gegebener Kurven mit Hilfe des Gelenkvierseits. *Zeitschrift für angewandte Mathematik und Mechanik* 3:13–19, 1923.

Find all four-bar linkages whose coupler curve passes through nine precision points.

C.W. Wampler, A.P. Morgan, A.J. Sommese: Complete Solution of the Nine-Point Path Synthesis Problem for Four-Bar Linkages. *Transactions of the ASME. Journal of Mechanical Design* 114(1): 153–159, 1992.

Formulation into Polynomial System

The 9-point problem was translated into

- a system of 4 quadrics and 8 quartics in 12 unknowns.
- Its total degree equals $2^4 4^8 = 2^{20}$.
- A 2-homogeneous Bézout number equals 286,720.
- Exploiting a 2-way symmetry leads to 143,360 solution paths.

At that time – early nineties – this was the largest polynomial system solved using numerical continuation methods.

Timings – Past and Present

Back Then: Tracking 143,360 solution paths in 12 variables took 331.9 hours of CPU time (about two weeks) on a IBM 3081 at the University of Notre Dame.

1,442 four-bar linkages were found

Computing various instances of the parameters with coefficient-parameter polynomial continuation requires only 1,442 paths to track. The number of real meaningful linkages ranged between 21 and 120.

Present: Using a personal cluster computer of 13 workers and one manager at 2.4 Ghz, running Linux, tracking 286,720 paths of a formulation in 20 variables takes about 14.1 hours.

The Theorems of Bernshteĭn

Theorem A: The number of roots of a generic system equals the mixed volume of its Newton polytopes.

Theorem B: Solutions at infinity are solutions of systems supported on faces of the Newton polytopes.

D.N. Bernshteĭn: **The number of roots of a system of equations.** *Functional Anal. Appl.*, 9(3):183–185, 1975.

Structure of proofs: First show Theorem B, looking at power series expansions of diverging paths defined by a linear homotopy starting at a generic system. Then show Theorem A, using Theorem B with a homotopy defined by *lifting* the polytopes.

Some References on Polyhedral Methods

I.M. Gel'fand, M.M. Kapranov, and A.V. Zelevinsky: **Discriminants, Resultants and Multidimensional Determinants**. Birkhäuser, 1994.

B. Huber and B. Sturmfels: **A polyhedral method for solving sparse polynomial systems**. *Math. Comp.* 64(212):1541–1555, 1995.

T.Y. Li.: **Numerical solution of polynomial systems by homotopy continuation methods**. In F. Cucker, editor, *Handbook of Numerical Analysis. Volume XI. Special Volume: Foundations of Computational Mathematics*, pages 209–304. North-Holland, 2003.

T. Gao and T.Y. Li and M. Wu: **Algorithm 846: MixedVol: a software package for mixed-volume computation**. *ACM Trans. Math. Softw.* 31(4):555–560, 2005.

T. Gunji, S. Kim, M. Kojima, A. Takeda, K. Fujisawa, and T. Mizutani: **PHoM – a polyhedral homotopy continuation method for polynomial systems**. *Computing* 73(4): 55–77, 2004.

T. Mizutani, A. Takeda, and M. Kojima: **Dynamic enumeration of all mixed cells**. *Discrete Comput. Geom.* to appear.

3 stages to solve a polynomial system $f(\mathbf{x}) = \mathbf{0}$

1. Compute the mixed volume (aka the BKK bound) of the Newton polytopes spanned by the supports A of f via a **regular mixed-cell configuration** Δ_ω.

2. Given Δ_ω, solve a generic system $g(\mathbf{x}) = \mathbf{0}$, using polyhedral homotopies. Every cell $C \in \Delta_\omega$ defines one homotopy

$$h_C(\mathbf{x}, s) = \sum_{\mathbf{a} \in C} c_{\mathbf{a}} \mathbf{x}^{\mathbf{a}} + \sum_{\mathbf{a} \in A \setminus C} c_{\mathbf{a}} \mathbf{x}^{\mathbf{a}} s^{\nu_{\mathbf{a}}}, \quad \nu_{\mathbf{a}} > 0,$$

   tracking as many paths as the mixed volume of the cell C, as s goes from 0 to 1.

3. Use $(1-t)g(\mathbf{x}) + tf(\mathbf{x}) = \mathbf{0}$ to solve $f(\mathbf{x}) = \mathbf{0}$.

Stages 2 and 3 are **computationally most intensive** ($1 \ll 2 < 3$).

A Static Distribution of the Workload

manager	worker 1	worker 2	worker 3
Vol(cell 1) = 5	#paths(cell 1) : 5		
Vol(cell 2) = 4	#paths(cell 2) : 4		
Vol(cell 3) = 4	#paths(cell 3) : 4		
Vol(cell 4) = 6	#paths(cell 4) : 1	#paths(cell 4) : 5	
Vol(cell 5) = 7		#paths(cell 5) : 7	
Vol(cell 6) = 3		#paths(cell 6) : 2	#paths(cell 6) : 1
Vol(cell 7) = 4			#paths(cell 7) : 4
Vol(cell 8) = 8			#paths(cell 8) : 8
total #paths : 41	#paths : 14	#paths : 14	#paths : 13

Since polyhedral homotopies solve a **generic** system $g(\mathbf{x}) = \mathbf{0}$,

we **expect** every path to take the same amount of work...

An academic Benchmark: cyclic n-roots

The system

$$f(\mathbf{x}) = \begin{cases} f_i = \sum_{j=0}^{n=1} \prod_{k=1}^{i} x_{(k+j) \bmod n} = 0, \quad i = 1, 2, \ldots, n-1 \\ f_n = x_0 x_1 x_2 \cdots x_{n-1} - 1 = 0 \end{cases}$$

appeared in

G. Björck: **Functions of modulus one on Z_p whose Fourier transforms have constant modulus** In *Proceedings of the Alfred Haar Memorial Conference, Budapest*, pages 193–197, 1985.

very sparse, well suited for polyhedral methods

Results on the cyclic n-roots problem

Problem	#Paths	CPU Time
cyclic 5-roots	70	0.13m
cyclic 6-roots	156	0.19m
cyclic 7-roots	924	0.30m
cyclic 8-roots	2,560	0.78m
cyclic 9-roots	11,016	3.64m
cyclic 10-roots	35,940	21.33m
cyclic 11-roots	184,756	2h 39m
cyclic 12-roots	500,352	24h 36m

Wall time for start systems to solve the cyclic n-roots problems, using 13 workers, with static load distribution.

Dynamic versus Static Workload Distribution

#workers	Static versus Dynamic on our cluster				Dynamic on argo	
	Static	Speedup	Dynamic	Speedup	Dynamic	Speedup
1	50.7021	–	53.0707	–	29.2389	–
2	24.5172	2.1	25.3852	2.1	15.5455	1.9
3	18.3850	2.8	17.6367	3.0	10.8063	2.7
4	14.6994	3.4	12.4157	4.2	7.9660	3.7
5	11.6913	4.3	10.3054	5.1	6.2054	4.7
6	10.3779	4.9	9.3411	5.7	5.0996	5.7
7	9.6877	5.2	8.4180	6.3	4.2603	6.9
8	7.8157	6.5	7.4337	7.1	3.8528	7.6
9	7.5133	6.8	6.8029	7.8	3.6010	8.1
10	6.9154	7.3	5.7883	9.2	3.2075	9.1
11	6.5668	7.7	5.3014	10.0	2.8427	10.3
12	6.4407	7.9	4.8232	11.0	2.5873	11.3
13	5.1462	9.8	4.6894	11.3	2.3224	12.6

Wall time in seconds to solve a start system for the cyclic 7-roots problem.

Design of Serial Chains

H.-J. Su and J.M. McCarthy: **Kinematic synthesis of RPS serial chains**. In the *Proceedings of the ASME Design Engineering Technical Conferences* (CDROM), Chicago, IL, Sep 2-6, 2003.

H.-J. Su, C.W. Wampler, and J.M. McCarthy: **Geometric design of cylindric PRS serial chains**. *ASME Journal of Mechanical Design* 126(2):269–277, 2004.

H.-J. Su, J.M. McCarthy, and L.T. Watson: **Generalized linear product homotopy algorithms and the computation of reachable surfaces**. *ASME Journal of Information and Computer Sciences in Engineering* 4(3):226–234, 2004.

Results on Design of Serial Chains

Bézout vs Bernshteĭn

Surface	Bounds on #Solutions			Wall Time	
	D	B	V	our cluster	on argo
elliptic cylinder	2,097,152	247,968	125,888	11h 33m	6h 12m
circular torus	2,097,152	868,352	474,112	7h 17m	4h 3m
general torus	4,194,304	448,702	226,512	14h 15m	6h 36m

D = total degree; B = generalized Bézout bound; V = mixed volume

Wall time for mechanism design problems on our cluster and argo.

- Compared to the linear-product bound, polyhedral homotopies cut the #paths about in half.
- The second example is easier (despite the larger #paths) because of increased sparsity, and thus lower evaluation cost.

Concluding Remarks

- To solve large polynomial systems in parallel we had to rethink the design of the original program.
- Scheduling of path tracking jobs leads to an almost optimal speedup, using dynamic load balancing.
- Still much work left to develop tools to process and certify the results, we need to consider also "quality up".

Software, source code and executables, available at

`http://www.math.uic.edu/~jan/download.html`