Computer Assisted Mechanics and Engineering Sciences, 11: 15–34, 2004.

A hybrid parallel approach to one-parameter nonlinear boundary value problems

Gábor Domokos[1] and Imre Szeberényi[2]
[1] *Department of Mechanics, Materials and Structures, Budapest University of Technology and Economics, H-1521 Budapest, Hungary. email: domokos@iit.bme.hu*
[2] *Department of Information Technology, Budapest University of Technology and Economics, H-1521 Budapest, Hungary. email: szebi@iit.bme.hu*

(Received February 25, 2002)

This paper presents a global algorithm for parallel computers, suitable to solve nonlinear boundary value problems depending on one parameter. Our method offers a mixture of path continuation and scanning. The former is well-known, the latter is a novel approach introduced a few years ago, capable to find *all* equilibria in a given domain. The hybrid method combines the speed of path continuation with the robustness and generality of scanning, offering a transition between the two methods which depends on the choice of some characteristic control parameters. We introduce the algorithms on a small example and test it on large-scale problems.

1. Introduction

This paper introduces a method for the global solution of nonlinear boundary value problems (BVPs). The basic idea is to combine path continuation with 'scanning', a novel approach introduced in [14]. As opposed to path continuation, the scanning algorithm can find *all* (even disconnected) equilibria in a given domain, however, it is extremely computation-intensive. The hybrid method presented here offers a transition between the two aforementioned methods, attempting to combine the advantages of fragile but fast continuation with those of slow but robust scanning. The mathematical background of our method is The Piecewise Linear (PL) algorithm (cf. [1]), combined with some simple ideas about ordinary differential equations (ODEs) and traditional shooting technique (so we will have to assume that the initial value problem (IVP) associated with the BVP is not stiff). This common platform makes the combination of scanning and continuation mathematically transparent and technically easy, also, it offers an almost continuous transition between the two approaches. However, it would be possible to combine algorithms with different background, e.g. to combine our scanning algorithm with AUTO [2], such a combination would offer much less flexibility but had all the advantages of AUTO. In this paper we concentrate on the 'homogeneous' approach and describe the resulting hybrid algorithm in detail.

Nonlinear BVPs are commonly resolved by incremental-iterative techniques, starting from the initial, trivial configuration and following the equilibrium path in small steps, based on extrapolation, cf. [25]. The error caused by the extrapolation is diminished by successive iterative steps, which are supposed to converge to the exact solution. Our algorithm adopts a different approach. We define a finite dimensional space (the Global Representation Space) into which the global solution of a nonlinear BVP can be embedded. We reduce the BVP to a nonlinear algebraic system by applying a forward integrator, subsequently we solve the equation system by discretizing the mentioned space

into simplices. The functions of our equation system can be piecewise linearly interpolated on this simplectic grid. The appeal of this approach is threefold:

- it does not contain iterative steps,
- it is capable of finding equilibria not connected to the trivial solution and
- it is highly suitable for parallelization.

Simultaneously – and quite naturally – the method has its weak sides. For the global results one has to pay with huge computation effort. This can be partially neutralized by the application of powerful parallel machines; our code is implemented under the PVM (Parallel Virtual Machine) system, which admits the utilization of heterogeneous networks. The tests confirmed that the parallel architecture can be utilized with high efficiency. These ideas are presented in Sec. 6. A further possibilty in the reduction of computation effort is the randomization of the algorithm which we will discuss in Subsec. 3.2.

The same basic ideas (reduction to algebraic system via integrator, simplectic decomposition) can also serve as a basis for an iteration-free path continuation code [4]. Assuming that one point of the solution curve is known, one can build a simplex around it and this simplex will contain a segment of the solution curve. Neighbouring simplices can be computed at minimal cost, so the path can be followed in both directions; the units of this algorithm are n-dimensional simplices. As opposed to the scanning version, this algorithm is very fast, producing one (approximate) BVP solution at the 'minimal cost' of *one* IVP integration. The disadvantage of continuation is that it can not find disconnected branches and it is not robust in the sense that failure of one integration process halts the algorithm. We will overcome the latter difficulties by introducing a generalized version, which, in exchange for somewhat higher computational costs, offers robustness. The unit of the robust continuation will be the n-dimensional cube.

The hybrid algorithm is a combination of the last described robust continuation with the previously mentioned randomized scanning. This combination requires that the essential parameters of the two algorithms are matched. However, there remain still several free parameters which admit the adjustment of the hybrid algorithm to the given problem and resources.

As already observed by several authors, ([7, 14]) discretization breeds spurious solutions. In our case a double discretization is applied: not only the embedding space but the ODE itself has to be discretized. Consequently we obtain two distinct classes of spurious solutions. Although we expect these solutions to vanish as the meshsize goes to zero, in many cases it is hard to tell whether a given solution is relevant or not. We will touch this subject only briefly, for more details we refer to [14].

In several sections we will apply an 'inverse' approach: rather then explaining first the general background and subsequently illustrate it on examples, we will demonstrate each step of the algorithm first on a very simple example (buckling of a cantilever beam under quasi-static axial load) and then generalize to arbitrary BVPs. It is impossible to explain all details of the algorithm based on one example, on the other hand, it is very hard to understand the essential parts without an illustrative example. Section 2 discusses the fundamental concepts, e.g. the Global Representation Space and the simplectic grid. Section 3 is devoted to the scanning algorithm (PSA), describing the simple (Subsec. 3.1) and the randomized (Subsec. 3.2) versions. Section 4 deals with the continuation, introducing first the serial version (SCA) in Subsec. 4.1 then the parallel continuation algorithm (PCA) in Subsec. 4.2. By matching the key parameters, the hybrid algorithm combines randomized scanning with parallel continuation; this is described in Sec. 5. The implementation, reference to examples, summary and conclusions are contained in Sec. 6. In particular, Subsec. 6.3 provides a qualitative analysis of large-scale computations and compares the predicted values with measured data from large tes-runs.

2. The basic concepts

2.1. The Global Representation Space: variables and functions

We will illustrate our method on the example of the axially compressed, uniform, elastic cantilever beam, illustrated in Fig. 1. The ODE describing the shape of the beam in terms of the slope α as a function of the arclength s was first described by Euler:

$$\alpha'' + P\sin(\alpha) + Q\cos(\alpha) = 0. \tag{1}$$

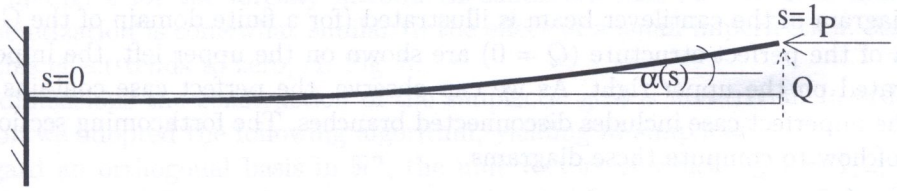

Fig. 1. The cantilever beam

The vertical force Q will be used as a small imperfection parameter which is *constant* during the loading process. The trajectories of this equation are uniquely determined by the three scalars $\alpha(0)$, $\alpha'(0)$ and P (the former ones being 'true' initial conditions, the latter one a parameter, Q is treated as a constant). However, not all trajectories are of interest for us, only those which meet the boundary conditions

$$\begin{array}{lrcl} a/ & \alpha(0) & = & 0, \\ b/ & \alpha'(1) & = & 0 \end{array} \tag{2}$$

expressing zero slope and zero curvature at the left and right end, respectively. For the time being, we ignore the far-end condition (2/b) and concentrate on (2/a). This condition eliminates one of the 'variable' initial conditions as a constant, so all trajectories which are candidates to meet the boundary conditions can be uniquely represented in the $[\alpha'(0), P]$ plane. Thus we managed to project the infinite-dimensional space of all geometrically possible configurations to a 2 dimensional space in such a way that the latter space is in a one-to-one correspondence with the set of candidate IVPs. The relevant BVP solutions can be regarded as a subset of these IVPs. We introduce the notation

$$\begin{array}{rcl} x_1 & \equiv & \alpha'(0), \\ x_2 & \equiv & P \end{array} \tag{3}$$

and the scalars x_i will be called the *global coordinates* (or *variables*) for this BVP, the plane (space)$[x_1, x_2]$ spanned by them will be called the *global representation space* (GRS) of the BVP. Since the BVP contains one parameter (P), the solutions will typically appear as one-dimensional manifolds, i.e. curves. Algebraically, these curves can be expressed as the solutions of the nonlinear equation corresponding to the far-end condition (2/b):

$$\alpha'(1) = f_1(\alpha'(0), P) = 0. \tag{4}$$

The far-end value of $\alpha'(1)$ is uniquely determined by our chosen variables $x_1 = \alpha'(0)$ and $x_2 = P$ via the *function* f_1, since the investigated ODE satisfies the conditions of Peano's Uniqueness Theorem. This is very often the case with ODEs related to mechanical problems; non-uniqueness arises, for example, in the case of strings. The application of our method requires not only unique dependence on initial data, we also need a non-stiff IVP, i.e. the IVP should not depend too sensitively on the initial conditions.

The solution of (4) appears on the $[\alpha'(0), P]$ plane (the GRS) as a set of curves. Our approach guarantees that whenever two solution curves intersect, the equilibria corresponding to those lines also coincide. We call such a diagram *topologically correct.*

On this simple example we introduced the basic concepts of our method. In the case of more complicated problems the GRS has more dimensions, however, it still remains a finite-dimensional space. In case of an n-dimensional GRS the variables x_i, $i = 1, 2, \ldots n$ are the non-constant initial conditions supplemented by the parameters, and the analogous equation system to (4) contains $(n-1)$ equations, defining the functions f_j, $j = 1, 2, \ldots n-1$. The $f_i = 0$ level sets determine $(n-1)$-dimensional hypersurfaces in the n-dimensional GRS, and the intersection of $n-1$ hypersurfaces results in – as in the case of the cantilever – 1-dimensional solution sets, i.e. curves. The assembly of those curves is called the global equilibrium path, or, the global bifurcation diagram. The global bifurcation diagram of the cantilever beam is illustrated (for a finite domain of the GRS) in Fig. 2: the equilibria of the perfect structure ($Q = 0$) are shown on the upper left, the imperfect ($Q = \epsilon$) case is illustrated on the upper right. As we can observe, the perfect case contains a bifurcation point while the imperfect case includes disconnected branches. The forthcoming sections investigate the problem of how to compute these diagrams.

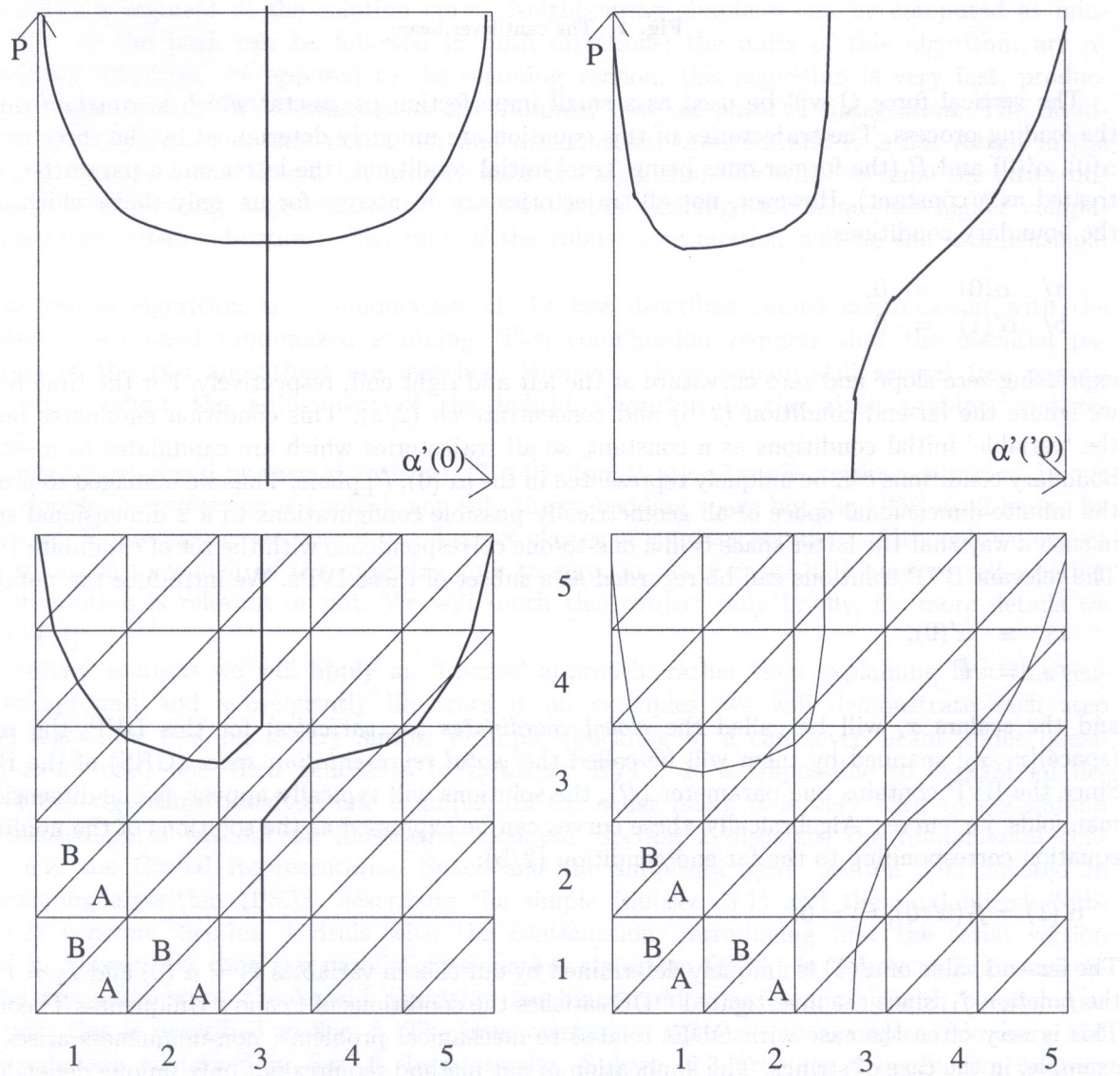

Fig. 2. The bifurcation diagram of the cantilever beam: perfect structure, $Q = 0$, (upper left) imprefect structure, $Q = \epsilon$, (upper right). Lower row: simplectic discretization of the GRS

2.2. Construction of the simplectic grid

In the previous section we established the global representation space (GRS) into which the global bifurcation diagram can be topologically correctly embedded. The computation of the diagram relies on the discretization of the GRS. The most natural way to discretize is to choose a simplectic grid (cf. [1]). An n-dimensional simplex is defined by $n+1$ points. (For example, a two-dimensional simplex is a triangle.) There are many ways to construct a simplectic grid. We choose the following method: in the first step we construct an orthogonal (cubic) grid and then, in the second step we subdivide each cube into $n!$ simplices. (For the sake of simplicity henceforth we refer to n-dimensional orthogonal objects as 'cubes' even if a linear transformation $\overline{x}_i = \lambda_i x_i, (i = 1, 2, ...n)$ is needed to transform them into cubes and even if $n = 2$.) The construction of the simplectic grid is illustrated in Fig. 2 for the already introduced cantilever example. Observe that the effect of simplectic discretization is somewhat similar to the effect of a small imperfection. As the grid gets more dense, this effect tends to zero, cf. Fig. 5.

In higher dimensions the construction of the simplectic grid is non-trivial. In order to simplify the subdivision we adopted the following algorithm, yielding $n!$ simplices:

Let us regard an orthogonal basis in $\Re^n$, the unit vectors of which $(\underline{e}_i, i = 1, 2, ..., n)$ span an n-dimensional unit cube. Let us denote a permutation (without repetition) of the first n natural numbers by $\nu_1, \nu_2, ...\nu_n$. The algorithm $\underline{r}_1 = 0, \underline{r}_{i+1} = \underline{r}_i + \underline{e}_{\nu_i}, (i = 1, 2, ...n)$ defines $n+1$ points in $\Re^n$. These points determine a simplex, containing all points $P(x_1, x_2, ...x_n)$ the coordinates of which satisfy the conditions

$$1 \geq x_{\nu_1} \geq x_{\nu_2} \geq ... \geq x_{\nu_n} \geq 0. \tag{5}$$

(If in any of these relations equality is satisfied then the point P is on the surface of the simplex, otherwise in the interior.) All permutations (without repetitions) define $n!$ simplices. These simplices fill the unit cube completely and without overlap: Take any point in (or on) the cube and arrange its coordinates by their magnitude. The order of the coordinates defines uniquely the simplex to which the selected point belongs.

2.3. Computation in one simplex

After having constructed the simplectic grid, the $n-1$ nonlinear functions are evaluated at the meshpoints. In the case of the cantilever example we have $n = 2$, thus a single fuction $\alpha'(1) = f_1(\alpha'(0), P)$ has to be evaluated at the meshpoints in the $[x_1, x_2] \equiv [\alpha'(0), P]$ GRS. This evaluation is realized by integrating the ODE (1) with initial conditions

$$\begin{aligned} \alpha(0) &= 0, \\ \alpha'(0) &= i\Delta_1, \\ P &= j\Delta_2 \end{aligned} \tag{6}$$

where Δ_1 and Δ_2 denote the meshsize in the $\alpha'(0)$ and P direction, respectively. One can apply any numerical scheme to integrate this equation. We remark that these schemes often yield spurious solutions for the BVP, so, the meshsize Δs for the arclength s has to be chosen as suitably small. The integration yields the desired values of f_1 at the meshpoints. (In the general case we obtain $n-1$ function values at each meshpoint.) In the case of the cantilever beam the function f_1 can be interpreted as a surface in the $[\alpha'(0), P, \alpha'(1)]$ space; the bifurcation diagram is the intersection of this surface with the GRS. Based on our just computed function values, the function f_1 can be piecewise linearly interpolated over the given domain by the C^0-continuous function f_1^L. Over each simplectic domain the function f_1^L is constructed as a linear combination of base functions, the coefficients of these latter are derived from the function values. The permutation $\nu_1, \nu_2, ...\nu_n$ defines $n+1$ points, thus a simplex of our grid. The value of the ith function f_i $(i = 1, 2, ...n-1)$ computed

at the jth vertex of the simplex will be denoted by f_{ij}, $(j = 1, 2, ...n + 1)$. In our example we have $n = 2$, and our single function f_1^L can be expressed as

$$f_1^L = f_{11} + (f_{12} - f_{11})x_{\nu_1} + (f_{13} - f_{12})x_{\nu_2}. \tag{7}$$

and in the general, n-dimensional case we have

$$f_i^L = f_{i1} + \sum_{j=1}^{n}(f_{i,j+1} - f_{ij})x_{\nu_j}. \tag{8}$$

In the cantilever problem, the intersection of f_1^L with the GRS is a polygonal line, this is the numerical approximation of the global bifurcation diagram. A simplex contains a segment of this line only if the sign of f_1 is not identical at the three vertices. If there are function values with different signs then the end-points of the line segment in the simplex can be determined by solving three equations systems, each with two unknowns; [14] describes the optimal solution technique.

3. The parallel scanning algorithm (PSA)

The previous section showed how one segment of the solution line can be obtained in a single simplex. Now we proceed to the description of how this process can be efficiently organized for a large number of simplices by utilizing the advantages of parallel computer architecture. The algorithm was implemented under the PVM (Parallel Virtual Machine) system, enabling the user to use individual computers on a distributed network as nodes of a virtual parallel machine. The program was designed according the so-called master-slave model.

3.1. Simple scanning

If we are interested in *all* solutions in a given domain of the GRS then we appply simple scanning, where *all* simplices are visited by the algorithm.

The cantilever example presented in Sec. 2 is very simple and the number of dimension is small ($n = 2$). A complex problem involves bigger GRS, and the number of dimension grows rapidly with the complexity of the problem. The CPU and memory requirements of the algorithm grow exponentially with the number of dimensions. In order to solve the equation system with prescribed precision we have to choose sufficiently small grid-size (Δ_i), so the number of cubes can be very large if the precision requirements are tight. Supposing that the number of points on each coordinate axis is N and the number of dimensions is n, the numbers of points where we have to evaluate each function will be N^n. This exponential expression can yield huge numbers, justifying the parallelization of the algorithm.

To design the parallel algorithm we have examined the simplex algorithm. The main observations are:

a. Every simplex is independent from the results of the calculations in the other simplices.

b. Since adjacent simplices have common vertices, the function values should not be re-computed for each simplex.

c. We assume that the computation time for the the functions f_i is not negligible, and it could be different at different points.

d. We expect solution points only in a few simplexes, and the most of the simplexes don't contain any solution points. (In the limit where the size of the simplices goes to zero we expect solution points "almost nowhere", on a subset of measure zero.)

Considering the first statement, i.e. that the computation in every simplex is independent suggests that the simplex could be the element of the parallelization, however the computation steps could be also parallelized in a simplex (e.g. solving the linearized functions and the n different equations of the simplex facets in parallel way). The main disadvantage of the parallelization inside the simplex is that the cost of the communication between two processes in PVM is very high.

The statement 'b' tells that the neighbour simplices have common vertices, so the same function values can be used. Statements 'c' and 'd' indicate that the load-balancing is also important in our case, however the capacity of the computers of the virtual machine can be different.

The implemented parallel program is based on a master-slave structure, where the master program distributes the phase space to smaller pieces (domains) and the slaves solve the equation system in this these domains. The major functions of the master program:

- reading the configuration files,
- starting and stopping the slaves,
- collecting the results from slaves,
- load-balancing.

The slave program essentially contains the serial version of the described algorithm, and solves the equations in the cubic domain given by the master. The values of the functions are computed once by the slave, when the slave gets a new domain from the master program. In this manner the function values are multiply computed only on the boundary points between the domains. To minimize the number of boundary points the master program tries to create domains with approximately equal orthogonal sizes.

We illustrate the scanning algorithm on the cantilever example. As shown in Fig. 2, the GRS has been divided into 5×5 'cubes'. Each cube will be identified by a two-digit number denoting the row and column. If we have only one processor then the GRS will not be subdivided and the cubes (with the simplices contained in them) will be computed in increasing order: $11, 12, 13, 14, 15, 21 \ldots, 54, 55$. As time unit we will choose the time needed to integrate (1) once, and we will neglect the time needed to solve the linear system. If we want to estimate the time needed for the computation, we must not forget that each function value is computed only once (we have only one slave process), so the total computation time will be $(5+1) \times (5+1) = 36$ units.

Assume now that we have 4 slaves and the GRS is subdivided by the master according to Fig. 3. This subdivision is 'optimal' in the sense that the number of domains and slaves is equal. In case of larger problems the memory capacity of the computer can limit the size of the domains. This subdivision can be imagined as constructing a *secondary orthogonal grid,* each 'secondary cube' containing in our case 3×3 'primary cubes'. Since the whole domain contains only 5×5 primary cubes, we can see only one 'full' 3×3 secondary unit, the remaining three are chopped off at the boundaries, resulting in two 2×3 and one 2×2 unit.

To find the computation time we draw a *flowchart,* the discrete horizontal axis will denote the time units. We must keep in mind that the computation time for one cube will differ: e.g. the first cube of each slave (11,14,31,34) will be computed in 4 units, while the second (12,15,32,35) in two units and there will be cubes which will be computed in one unit. Table 1 shows the flowchart for the computation. Needless to say, the flowchart will be identical for the imperfect case, since simple scanning is insensitive to the structure of the bifurcation diagram. Figure 4 displays the flowchart graphically, jointly with the flowcharts of other algorithms, offering a good visual comparison. In order to compare different methods not only on a visual, but a more exact basis, we introduce a few characteristic parameters:

1. The number of all straight solution segments in the domain is denoted by L.
2. The number of simplices computed by k slaves is denoted by S_k.
3. The number of solution segments computed by n slaves is denoted by L_k.
4. The time of computation with k slaves (measured in the above defined discrete units) is denoted by T_k.

5. The speed of computation is defined by $V_k = L_k/T_k$.
6. The reliability of the algorithm is defined as $R_k = L_k/L$.
7. The efficiency of the algorithm is defined as $E_k = L_k/S_k$.
8. The speedup of the algorithm is defined as $U_k = V_k/V_1$.
9. The speedup factor of the algorithm is defined for $k > 1$ as $F_k = (U_k - 1)/(k - 1)$.

Table 1. Simple scanning: flowchart with 4 slaves

TIME	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SLAVE 1	11	→	→	→	12	→	13	→	21	→	22	23				
SLAVE 2	14	→	→	→	15	→	24	→	25							
SLAVE 3	31	→	→	→	32	→	33	→	41	→	42	23	51	→	52	53
SLAVE 4	34	→	→	→	35	→	44	→	45	54	→	55				

Before evaluating these parameters for the scanning algorithm, a few comments might be helpful. Inspection of Fig. 2 reveals that in the case of the perfect ($Q = 0$) problem $L = 24$, in case of the imperfect problem $L = 22$. Since the scanning algorithm computes all simplices, we have $S_k = 50$ in both cases, independently of the number k of slaves. For the same reason, we have $L_k = L$ in both cases. The relevant data are summarized in Table 2. The slight difference between the perfect and imperfect case is due to the slightly different number L of solution segments. We can observe that this method is 100 percent reliable. If we compute larger domains in higher dimensions, the efficiency of the simple scanning approaches zero.

Table 2. Simple scanning: characteristic parameters

	V_1	E_1	R_1	V_4	E_4	R_4	U_4	F_4
PERFECT PROBLEM	0.67	0.48	1.00	1.50	0.48	1.00	2.24	0.41
IMPERFECT PROBLEM	0.61	0.44	1.00	1.37	0.44	1.00	2.25	0.42

3.2. Randomized scanning

The scanning algorithm computes systematically all simplices. This search can be very time consuming, especially if the dimension of the GRS is high. On the other hand, the scanning algorithm offers 100 percent reliability in the sense that only solutions that can be missed are closed curves where the distance of the curves is smaller than the gridsize. Is it possible to maintain this level of reliability while accelerating the search? In some sense the answer is positive: if, instead of systematic, we compute the simplices in a random order, we can hope to find the "bulk" of solution segments sooner. Nevertheless, in order the same reliability, we have to compute all solution segments, so the total computation cost will not decrease. What one can hope for is to improve the distribution of computed solution segments as a function of time.

There are many ways to randomize the computation. The most straightforward option is to regard the *secondary cubes* as units. In this case all characteristic parameters of the systematic and randomized scanning will be identical. In the given example the secondary grid does not have enough units in the given domain to illustrate the effect of randomization (cf. Fig. 3), so let us regard the primary 'cubes' as units in order to illustrate the basic features. We will assign an *index* to each cube, this index will indicate the number of *adjacent cubes which have not been computed previously.* We call two cubes adjacent (neighbors) if they have common $(n - 1)$-dimensional faces, e.g. in our

example the cubes in the middle have 4 neighbors, the cubes in the corners have 2 neighbors. In the initial configuration the index table is determined solely by geometry: see the first matrix in Table 3. Assume 4 slaves as before: in the first step 4 cubes with maximal indices are selected and computed, the indices of these cubes drop to -1, and the index table is updated accordingly. In the second step again 4 cubes with maximal indices are selected, cf. Table 3/b, and this process is carried on as long as non-negative indices remain. The basic idea of this method is that the larger the 'connected' domain, the larger the chance to find new solutions. This algorithm requires that the index table is stored and updated, this can impose limitations on the size of the domain.

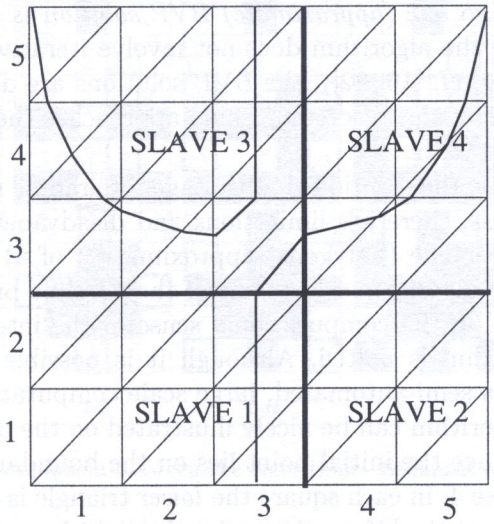

Fig. 3. The subdivision of the GRS in case of 4 slaves: secondary grid with 3×3 units. Due to the small size of the 5×5 total domain we can observe different domains with 2×2 and 2×3 units

Table 3. Radomized scanning: index table on the primary 5×5 grid for the first three steps of computation

2	3	3	3	2
3	4	4	4	3
3	4	4	4	3
3	4	4	4	3
2	3	3	3	2

2	3	3	2	2
3	4	2	−1	2
3	2	−1	1	3
2	−1	1	−1	2
2	2	3	2	2

2	2	3	2	2
1	−1	1	−1	1
−1	0	−1	0	−1
1	−1	0	−1	1
2	1	−1	1	2

4. The continuation algorithm

In order to do path continuation we need the GRS location of at least one solution point, at least approximately. Most commonly, this is a trivial configuration; in our example this corresponds to the unloaded ($P = 0$) cantilever. In the imperfect, $Q > 0$ case the initial configuration is less trivial, however, it suffices to assume that it is close enough to the $Q = 0$ perfect system. Our goal is to continue the branch (path) defined by this point.

4.1. The serial continuation algorithm (SCA)

The serial continuation algorithm has been introduced in [4], here we describe a slightly different version. We construct our primary orthogonal grid in such a way that the initial solution lies inside one specified simplex. In the cantilever example this is the simplex $13/A$, cf. Fig. 2. The specified simplex in which the solution point lies will contain one solution segment.

Our next goal is to define the *next* simplex in each direction. This is accomplished by identifying the *entering* and *exit* surfaces and points. In our case the lower, horizontal side of thesimplex $13A$ will be called the entering surface and the upper side the exit surface, the points where the straight solution segment intersects these faces will be called the entering point and exit point, respectively. The continuation is defined by identifying the exit face of the previous simplex with the entering face of the next one, and similarly, the exit point of the previous simplex with the entering point of the next one.

If two simplices have one common face, then each of them contains only one vertex which is not contained in the other. This means that the computation of the subsequent simplex includes only one forward integration, so *one (approximate) BVP solution is obtained at the minimal 'cost' of one IVP integration.* Since the algorithm does not involve iterative steps (disregarding possible iteration *inside* the IVP solver, cf. [16, 22]), the BVP solutions are delivered by a direct recursion. As the meshsize approaches zero, this discrete formula approaches the differential equation defining the BVP solution curve in the GRS.

Beside its theoretical beauty, the mentioned advantages guarantee that this algorithm is very fast and very efficient. Nevertheless, there are limitations and disadvanatges. The principal limitation is that this method can deliver one (polygonal approximation of a) curve, bifurcation points are 'ignored' in the sense that the algorithm selects one of the possible branches effectively at random. This algorithm is not 'robust' in the computational sense: if the integration (function evaluation) fails at one point, the algorithm is halted. Although it is possible to start it again from some intermediate point, this makes semi-automated, large scale computations very cumbersome.

The serial continuation algorithm can be nicely illustrated on the cantilever example: the continuaion will be unidirectional since the initial point lies on the boundary of the investigated domain. The flowchart is shown in Table 4, in each square the *lower* triangle is denoted by 'A', the upper one by 'B', cf. Fig. 2. Observe that at the bifurcation point the right branch is selected by the algorithm, this is determined by the direction of the diagonals on the simplectic grid. Only one slave is applied, this is an essentially serial algorithm where parallelization does not make much sense. (To compare the SCA with the parallel methods, cf. Fig. 4 for the visual comparison of flowcharts.) This is also apparent from Table 6 where the essential parameters of the algorithm are summarized; we can see that the speedup factor is zero, the application of further slaves does not accelerate the process. The flowchart for the imperfect problem, shown in Table 5 is slightly different: there is no bifurcation point in this problem and the number of computed solution segments is slighly different as well.

We remark that the original serial continuation algorithm, introduced in [4] did not operate on a fixed simplectic grid, the subsequent simplex was identified by reflecting one vertex of the previous simplex with respect to the exit face.

Table 4. Serial continuation: flowchart for the perfect problem

TIME	1	2	3	4	5	6	7	8	9	10	11	12	13
SLAVE 1	13A	→	→	13B	23A	23B	33A	34B	34A	35B	45A	45B	55A

Table 5. Serial continuation: flowchart for the imperfect problem

TIME	1	2	3	4	5	6	7	8	9	10	11
SLAVE 1	13A	→	→	13B	23A	24B	34A	35B	45A	45B	55A

Table 6. Serial continuation: characteristic parameters

	V_1	E_1	R_1	V_4	E_4	R_4	U_4	F_4
PERFECT PROBLEM	0.85	1.00	0.45	0.85	1.00	0.45	1.00	0.00
IMPERFECT PROBLEM	0.82	1.00	0.41	0.82	1.00	0.41	1.00	0.00

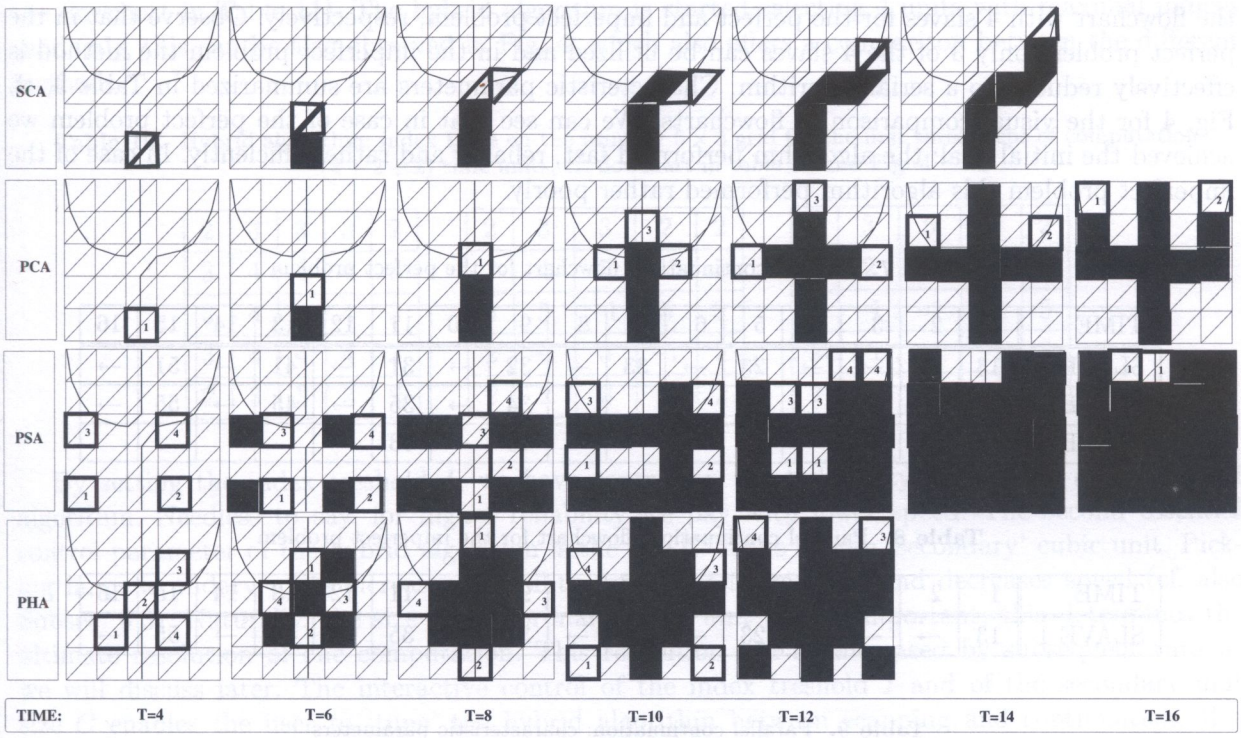

Fig. 4. Graphical comparison between the algorithms: visualization of the flowcharts

4.2. The parallel continuation algorithm (PCA)

In the previous subsection we saw that serial continuation is very efficient but rather unreliable. Our goal is to increase the reliablity and robustness at the cost of some losses in efficiency. The basic idea is to choose the cube (rather than the simplex) as the unit of continuation. As we described, the simplex can have only one entry and one exit point, while the cube can have n entry and n exit points, where n is the dimension of the GRS. This property enables the cube-based algorithm to handle bifurcation points, so it makes now sense to apply multiple slave processes which may follow the branches originating at the bifurcation point. As the slave process follows one branch, there will be cubes where the number of solution segments will be less than $n!$, which is the number of simplices in the cube. This implies that the efficiency of this algorithm will be less than 1. Robustness and reliability can be further increased by using 'secondary cubes' as units, defined by a secondary grid (cf. Fig. 3), needless to say that the larger the cubic units, the smaller the efficiency.

The parallel continuation algorithm is (as opposed to the serial one) capable of finding 'slighly' disconnected branches. 'Slightly' means that the disconnected branch is close enough to intersect the same cubic unit. Increasing the size of the unit increases the chances to 'catch' disconnected branches. The serial continuation code was unable to handle bifurcation points since the piecewise linear approximation dissolves bifurcation points into disconnected branches. Nevertheless, these branches are only 'minimally' disconnected, since already the smallest (primary) cubic unit is sufficient to capture the disconnected branch, so the parallel continuation algorithm handles bifurcation points more efficiently than the serial version (however, there is no guarantee that each bifurcation point will be captured).

The previously presented algorithms performed almost identically on the perfect and imperfect cantilever problem. If we choose the primary cubes as continuation units, the parallel continuation algorithm will display radically different behaviour: it will capture the left branch in the perfect case, however, it will fail to identify it in the imperfect case. Incresing the size of the cubic unit would enable the algorithm to solve the imperfect case completely as well. Tables 7 and 8 show

the flowchart with 4 slaves for the perfect and imperfect problem, respectively. Observe that in the perfect problem only 3 of the 4 slaves can be utilized and in the imperfect problem the method is effectively reduced to a serial algorithm. Characteristic parameters are summarized in Table 9, cf. Fig. 4 for the visual comparison of flowcharts. We can see that in case of the perfect problem we achieved the initial goal: the algorithm performed fast, reliably and rather efficiently. In case of the imperfect problem this algorithm performed rather poorly.

Table 7. Parallel continuation: flowchart for the perfect problem

TIME	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SLAVE 1	13	→	→	→	23	→	33	→	32	→	31	→	41	→	51	→
SLAVE 2									34	→	35	→	45	→	55	→
SLAVE 3									43	→	53	→				

Table 8. Parallel continuation: flowchart for the imperfect problem

TIME	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SLAVE 1	13	→	→	→	23	→	24	→	34	→	35	→	45	→	55	→

Table 9. Parallel continuation: characteristic parameters

	V_1	E_1	R_1	V_4	E_4	R_4	U_4	F_4
PERFECT PROBLEM	0.86	0.92	1.00	1.50	0.92	1.00	1.75	0.25
IMPERFECT PROBLEM	0.56	0.64	0.41	0.56	0.64	0.41	1.00	0.00

5. The parallel hybrid algorithm (PHA)

Our goal is to merge the scanning (PSA) and continuation (PCA) algorithms in such a way that the resulting hybrid method (PHA) inherits the robustness and reliability of scanning together with the speed and efficiency of continuation. The basic idea is to combine *randomized* scanning with *parallel* continuation. The index table associated with the randomized scanning did not store data on the identified solution segments. Recall that the maximal index in n dimensions is $2n$. In the hybrid algorithm we will modify the index table in the following manner: if solutions segments are found in a cubic unit, then the indices of those neighbor cubes to which the segments connect will be increased to $2n+1$ (unless they have been computed beforehand). These neighbor units are bound to contain new solution segments, and since they will have the highest index values, they will be computed first. In this way, as long as the branch can be followed, the hybrid algorithm operates essentially as the parallel continuation algorithm. If the branch followed by the slave process is terminated for any reason (e.g. the boundary of the domain is reached) then the master will assign either the continuation of another branch, or, if no such task is available, randomized scanning will be resumed by selecting the highest index.

Since the original randomized scanning method did not gather information on the solution segments, it had to be continued until he last of the cubic units was computed, which is equivalent to have only negative indices in the index table. In case of the hybrid algorithm we can determine what is the maximal size of the branch which we can ignore and set a treshold I for the minimal index value accordingly. For example, if we decide to ignore branches which fit into one cubic unit then the $I = 0$, since this way only 'isolated' non-computed cubic units are permitted. If we set $I = 1$, then a non-computed branch with maximal size will fit into two neighbor cubic units. Table 10 illustrates the index tables in the perfect cantilever problem, for the first three steps of computation (the first step is equivalent to 4 time units, the second and third steps to 2 time units, cf.

the flowchart in Table 11). The hybrid algorithm is started selecting 4 units with maximal indices randomly, this can be also observed in Fig. 4, which also offers a comparison between the different methods.

Table 10. Hybrid algorithm: index table on the primary 5×5 grid for the first three steps of computation $(4+2+2)$ time units, cf. flowchart in table 11 and Fig. 4

2	3	3	3	2
3	4	4	4	3
3	4	4	4	3
3	4	4	4	3
2	3	3	3	2

2	3	3	2	2
3	4	5	−1	5
3	5	−1	5	3
2	−1	5	−1	2
2	2	3	2	2

2	3	5	2	2
3	2	−1	−1	5
5	−1	−1	−1	5
2	−1	−1	−1	2
2	2	5	2	2

By setting the index treshold I, we have control on the 'reliability' parameter of the hybrid algorithm. Needless to say, for higher reliability we pay with lower speed. The second essential control parameter of the hybrid algorithm is the size C of the chosen 'secondary' cubic unit. Picking large secondary units increases reliability (at constant treshold) and decreases speed (cf. also Subsec. 6.3). Naturally, the size of the primary cubic unit is also important, this determines the ultimate resolution of one computation. This resolution can be increased by subsequent runs as we will discuss later. The interactive control of the index treshold I and of the secondary unit size C enables the user to 'tune' the hybrid algorithm between scanning and continuation. If a solution segment is identified in one cubic unit then the function values on the surface of this unit are passed on to the master programme. When the master assigns the connecting unit to the next available slave, the relevant function values are passed to the slave so these do not have to be recomputed.

The determination of the speed of the hybrid algorithm is rather delicate, because of the randomization we can only identify the *expected value,* or, if needed, the distribution. To illustrate the cantilever example, we picked a 'typical' initial configuration. (For the description of large-scale computations, cf. Subsec. 6.3.) If we start from this configuration, the flowcharts for the perfect and the imperfect problem with $I = C = 1$ will be identical. In Table 12 of the characteristic parameters the speed is represented by the expected value, which is slightly less then the speed illustrated in Table 11. Observe on the flowcharts that the hybrid algorithm can utilize the 4 slaves very efficiently, because it starts to compute the 4 branches from the bifurcation point simultaneously, rather than 'arriving' on one branch and 'exiting' on three, as the parallel continuation algorithm does, cf. Fig. 4 for visual comparison of the flowcharts.

Table 11. Hybrid algorithm with $I = C = 1$: flowchart for the perfect and imperfect problem

TIME	1	2	3	4	5	6	7	8	9	10	11	12
SLAVE 1	22	→	→	→	43	→	53	→	11	→	→	→
SLAVE 2	33	→	→	→	23	→	13	→	15	→	→	→
SLAVE 3	44	→	→	→	34	→	35	→	45	→	55	→
SLAVE 4	24	→	→	→	32	→	31	→	41	→	51	→

Table 12. Hybrid algorithm with $I = C = 1$: characteristic parameters

	V_1	E_1	R_1	V_4	E_4	R_4	U_4	F_4
PERFECT PROBLEM	0.52	0.67	1.00	1.64	0.67	1.00	3.21	0.73
IMPERFECT PROBLEM	0.51	0.64	1.00	1.61	0.65	1.00	3.15	0.72

6. Summary and comparison of the algorithms

6.1. Comparison of the algorithms

All described algorithms are based on the simplectic decomposition of the GRS, described in Sec. 2. It is not easy to compare the algorithms, however, compiling the tables related to the cantilever example might be helpful. Table 13 summarizes the data for the perfect problem, Table 14 for the imperfect case. The cantilever example is far to small for any quantitative conclusions. However, some qualitative trends are already visible in this model problem. We can observe that the efficiency of the PSA is rather small: as the dimension and size of the problem grows, both efficiency and speed of the PSA approach zero. This is not true for the continuation and the hybrid algorithms, however, even parallel continuation works reliably only for connected solutions. Our conclusion is that if we are interested in disconnected branches as well, the hybrid algorithm is the only feasible choice. The other main attraction of the PHA is that it shows the highest speedup factor, indicating that this algorithm can most efficiently utilize the advantages of parallel architecture. The algorithms can be nicely compared in Tables 15 and 16 which have been compiled from the flowcharts of the perfect and imperfect problems, respectively. The flowcharts can be visually compared in Fig. 4.

Table 13. Comparsion of characteristic parameters: perfect problem

	V_1	E_1	R_1	V_4	E_4	R_4	U_4	F_4
PSA	0.67	0.48	1.00	1.50	0.48	1.00	2.24	0.41
SCA	0.85	1.00	0.45	0.85	1.00	0.45	1.00	0.00
PCA	0.86	0.92	1.00	1.50	0.92	1.00	1.75	0.25
PHA	0.52	0.67	1.00	1.64	0.67	1.00	3.21	0.73

Table 14. Comparsion of characteristic parameters: imperfect problem

	V_1	E_1	R_1	V_4	E_4	R_4	U_4	F_4
PSA	0.61	0.44	1.00	1.37	0.44	1.00	2.25	0.42
SCA	0.82	1.00	0.41	0.82	1.00	0.41	1.00	0.00
PCA	0.56	0.64	0.41	0.56	0.64	0.41	1.00	0.00
PHA	0.56	0.64	1.00	1.67	0.65	1.00	2.98	0.66

Table 15. Comparison of flowcharts for the perfect problem

TIME	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
PSA1	11	→	→	→	12	→	13	→	21	→	22	23				
PSA2	14	→	→	→	15	→	24	→	25							
PSA3	31	→	→	→	32	→	33	→	41	→	42	23	51	→	52	53
PSA4	34	→	→	→	35	→	44	→	45	54	→	55				
SCA1	13A	→	→	13B	23A	23B	33A	34B	34A	35B	45A	45B	55A			
PCA1	13	→	→	→	23	→	33	→	32	→	31	→	41	→	51	→
PCA2									34	→	35	→	45	→	55	→
PCA3									43	→	53	→				
PHA1	22	→	→	→	43	→	53	→	11	→	→	→				
PHA2	33	→	→	→	23	→	13	→	15	→	→	→				
PHA3	44	→	→	→	34	→	35	→	45	→	55	→				
PHA4	24	→	→	→	32	→	31	→	41	→	51	→				

Table 16. Comparison of flowcharts for the imperfect problem

TIME	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
PSA1	11	→	→	→	12	→	13	→	21	→	22	23				
PSA2	14	→	→	→	15	→	24	→	25							
PSA3	31	→	→	→	32	→	33	→	41	→	42	23	51	→	52	53
PSA4	34	→	→	→	35	→	44	→	45	54	→	55				
SCA1	13A	→	→	13B	23A	24B	34A	35B	45A	45B	55A					
PCA1	13	→	→	→	23	→	24	→	34	→	35	→	45	→	55	→
PHA1	22	→	→	→	43	→	53	→	11	→	→	→				
PHA2	33	→	→	→	23	→	13	→	15	→	→	→				
PHA3	44	→	→	→	34	→	35	→	45	→	55	→				
PHA4	24	→	→	→	32	→	31	→	41	→	51	→				

6.2. Additions and generalizations

This paper described only the backbone of the algorithms, nevertheless, responding to the needs of concrete problems many additions and generalizations have been implemented, we mention only the more substantial ones.

The PSA has been extended to functions with discontinuities in [12]. Similar difficulties are treated in case of the SCA in [21]. As mentioned before, due to the double discretization (physical space and GRS) two distinct classes of spurious solutions emerge, these are described in more detail in [14]. One efficient method to fight the GRS-related spurious solutions is the idea of *zooming,* where the density of the orthogonal grid is multiplied in the units containing solution segments after the first run. This method has been applied successfully in [9], [20] and [5] and is illustrated for the cantilever example in Fig. 5. In the second run only the units with dense grid are computed. Zooming can be combined both with the PSA and the PHA.

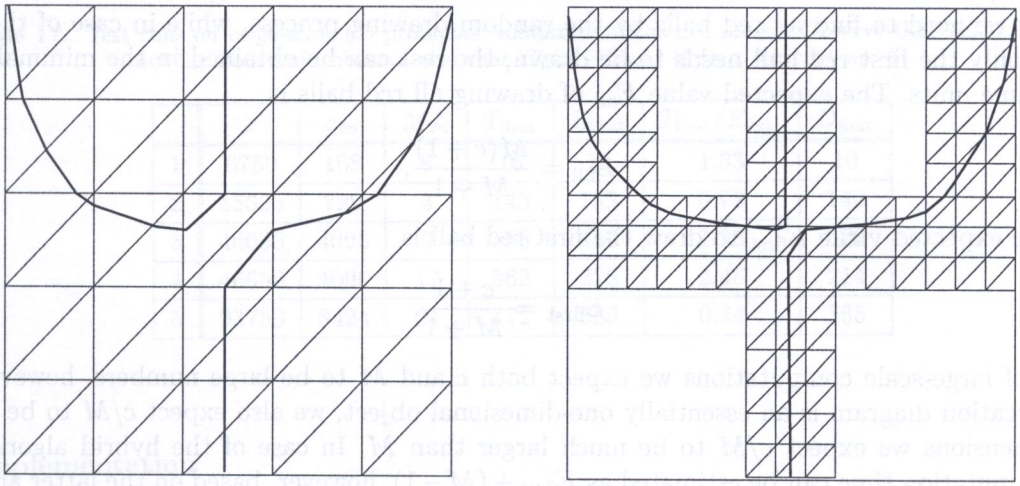

Fig. 5. The zooming algorithm: grids for two subsequent computations

In some cases one is interested not in the bifurcation diagram itself but in the corresponding *stability chart,* which contains the locus of extrema of the bifurcation diagram as a function of a new parameter. We introduced the recursive version of the PSA in [11], capable to reduce such problems with k additional parameters to problems in k additional dimensions. The recursive PSA can compute the stability diagrams as solutions of a well-defined $n+1$ dimensional problem. The algorithm is a recursion, because in order to evaluate one of the function values, the PSA algorithm

calls itself as a subroutine in a lower dimensional space. The recursion can be used both for the PSA and the PHA.

6.3. Large-scale computations

The cantilever example, used thoroughout the paper, is appropriate to illustrate the mechanisms of the computation and the fine details of the algorithms, however, it is much too small to give an idea how these methods behave in case of real, large-scale computations. Nevertheless, the main motivation behind the construction of these algorithms is to perform large-scale computations, so a general decsription should not be avoided.

The most interesting question is to determine (at least approximately) the *ratio of speed* between the PSA (scanning) and PHA (hybrid) algorithms. According to Table 13, in the case of the perfect cantilever problem and with the application of 4 slaves, this ratio was $r = 1.64/1.50 = 1.0953$. This *indicates* that the hybrid method is faster, but provides no basis for a quantitative estimate for large-scale problems.

To simplify the discussion we will assume for both algorithms that the investigated domain is n-dimensional, containing $c = N^n$ *secondary* cubes of equal size. (The number of these larger units is most relevant if we are interested in the speed ratio r. In the cantilever problem we had $c = 2^2 = 4$ (non-equal) secondary units in the case of the PSA and $c = 5^2 = 25$ (equal) units in case of the PHA computation.) To simplify the comparison even further, we will assume that in the investigated domain the bifurcation diagram consists only of *one single connected component* , and this component intersects M secondary cubic units (out of $c = N^n$). In order to obtain the speed, we have to estimate the time necessary to compute all secondary cubes containing solution segments. Since we assumed these units to be equal, the computation time can be measured approximately by taking the computation time of one secondary cube as a unit.

Since no information is available on the distribution of the M units containing solution segments, we will regard them as being at random locations, so the computation is equivalent of drawing red and white balls from an urn without replacement, the total number of balls is $c = N^n$ out of which M are red, $c - M$ are white. The fundamental difference between the algorithms is that in case of scanning we need to find all red balls by the random drawing process, while in case of the hybrid method only the first red ball needs to be drawn, the rest can be obtained in the minimal time of $M - 1$ time units. The expected value E_{all} of drawing all red balls is

$$E_{\text{all}} = \frac{M(c+1)}{M+1},$$

while the expected value E_{first} to draw the first red ball is

$$E_{\text{first}} = \frac{c+1}{M+1}.$$

In case of large-scale computations we expect both c and M to be large numbers, however, since the bifurcation diagram is an essentially one-dimesional object, we also expect c/M to be large, in high dimensions we expect c/M to be much larger than M. In case of the hybrid algorithm the total computation time can be estimated as $E_{\text{first}} + (M-1)$, however, based on the latter argument, the $M - 1$ time units (needed to complete the diagram following the identification of the first cube containing solution segments) can be neglected compared to the $E_{\text{first}} \approx c/M$ time units necessary to identify the first cube. Thus, we can estimate the speed ratio of the two methods based on the ratio of the expected values obtained from the urn model, and this yields

$$r \approx E_{\text{all}}/E_{\text{first}} = M. \tag{9}$$

In plain English this means that in case of large-scale computations the speed factor gained by the application of the hybrid algorithm is roughly equal to the number of secondary units containing

solution segments. In real computations, particularly in high dimensions, this speed factor can be a very large number, thus the hybrid method can offer an advantage of several orders of magnitude. Naturally, this implies that smaller secondary units are of advantage if we apply the hybrid method. As the size of the secondary units increases, the scanning and hybrid methods approach each other in speed. Also, the computed speed ratio decreases if the bifuraction diagram contains more than one connected component. Ultimately, in case of many scattered small components, the two methods become equivalent.

One important feature of the hybrid method is that it applies the index table described in Subsec. 3.2. The computation is carried out always on the units having maximal current index. In the initial configuration, before starting the computation, the units *in the interior* of the n-dimensional domain will have maximal indices $2n$, so the random selection will be carried out only among these interior units. The ratio of the interior units to all units is $c_{2n}/c = (N-1)^n/N^n = ((N-1)/N)^n = 1-(1/N)^n$ and it approaches zero at $N = constant$ as $n \to \infty$. This offers an additional speed advantage for the hybrid method.

We can conclude that the speed advantage of the hybrid method increases both with the number of discretization units and with the number of dimensions. We carried out test runs on a 6D problem ([6]) to verify the results from the urn model for the hybrid algorithm. Table 17 summarizes the results; the first column contains the serial number of the test run, the second column the number c of all secondary units, the third column the number c_{2n} of units with maximal index $2n$, the fourth column the number M_{2n} of units with index $2n$ containing solution segments, the fifth column the time T_{first} (in secondary units) of the identification of the first solution segment, the sixth column the value E_{first}, predicting T_{first}, the seventh column provides the ratio $T_{\text{first}}/E_{\text{first}}$ of the actual and estimated value. We also computed the standard deviation D_{first}, this is given in the eigth column. As we can observe, the ratio $T_{\text{first}}/E_{\text{first}}$ is not very far from unit, and we always have $|T_{\text{first}} - E_{\text{first}}| < D_{\text{first}}$, which also confirms the validity of our approach. The data in Table 17 shows that in these 6D test-runs the actual gain by using the hybrid algorithm was huge: the ratio c/T_{first} estimates roughly the speed ratio between the hybrid and the scanning algorithm and this value varies between 83 and 545. This is certainly much larger then the estimate in Eq. (9) predicts, however, (9) did not consider the effect of the index table discussed in the previous paragraph.

Table 17. Test runs on large-scale 6D problems: validation of the urn model. Observe that $T_{\text{first}}/E_{\text{first}}$ is not far from unit, observe also that $|T_{\text{first}} - E_{\text{first}}| < D_{\text{first}}$ (except no. 4)

	c	c_{2n}	M_{2n}	T_{first}	E_{first}	$T_{\text{first}}/E_{\text{first}}$	D_{first}
1	6750	108	8	16	12	1.33	10
2	15625	729	3	143	183	0.78	140
3	46656	4096	9	558	410	1.36	370
4	46656	4096	17	562	228	2.46	214
5	93750	8424	21	172	383	0.44	365

6.4. Implementation

As mentioned before, all parallel versions of the algorithm have been implemented under the PVM (Parallel Virtual Machine) system [17] using a master-slave model.

The load-balancing is provided by the master, because the GRS is usually divided into more domains than the number of processors. When the computation in any domain has been finished, the master sends the next domain to the next free slave. In this way the faster processors will get more jobs then the slower processors.

The parallel program was developed on heterogeneous environment. We have used for development different architectures (HP 9000, VAX-750, SUN IPC). This environment is good for testing,

but performance measurement is quite difficult since the different computers have different processors. However, we could test the application, and the load-balancing as well on an IBM SP1 computer with 8 processors and 3 RS6000/580 computers connected by Ethernet with TCP/IP protocol, also on the 140-node IBM SP2 at the Cornell Supercomputing Center and recently on a 20-node parallel machine constructed from Pentium III PCs. In all these environments we measured almost linear speedup for the PSA. Comparing the speed of the PSA and the PHA on large scale problems is still under way.

6.5. Visualization of results

One of the advantages of our GRS-based mathematical model is that it admits an easy and interactive visualization of the results. The basic idea is to have two windows open: one of them contains the bifurcation diagram in the GRS, and one can pick points on this diagram interactively. When picking the point, the GRS coordinates are identified and based on them, via the integration of an IVP, the corresponding physical shape is computed instantaneously and displayed in the other window. For more details see [10]. This visualization system has now been implemented under IBM's Data Explorer and can be used to view 3 dimensional physical objects with corresponding bifurcation diagram in arbitrary dimension.

6.6. Conclusions

We presented a new algorithm, combining the advantages of path continuation and scanning. This combination can be realized because both versions (the continuation and the scanning algorithm) work in the same mathematical framework, embedding the BVP in the global representation space (GRS) and discretizing the GRS into simplices. The mathematical background is the piecewise linear algorithm described in [1]. Although this common platform makes the combination of scanning and continuation very easy, it might be worth to combine the PSA or its randomized version with other type of continuation codes, e.g. AUTO (cf. [2]) which can handle stiff problems as well. Also, combination with DSTOOL [18] is an interesting project, the current version of DSTOOL includes the possibility to scan a subspace for 'seeds' of branches which can be followed subsequently. Our algorithm is different because the random scan is extended to the complete space and by adjusting the control parameters one has a continuous choice between complete scan and simple continuation.

On a small example, the buckling of a cantilever, we presented the different versions of the algorithm in detail and defined characteristic parameters to measure performance. These parameters include the speed, the efficiency, the reliablility and speedup factor associated with the given algorithm. Evaluating these parameters indicate that the suggested hybrid algorithm is efficient, fast and reliable in a wide range of applications. We provided a simple probabilistic model to estimate the performance of the hybrid algorithm in large-scale computations. The test-runs on a 6D problem validated our model and both the theoretical and the numerical results indicate that the hybrid method can accelerate the computation by *several orders of magnitude*. (In our test we predicted and measured accelration factors between 83 and 545.) These invetsigations confirmed that the excess speed provided by the hybrid methods growth with the size and dimension of the task. Based on these results we hope that by using the PHA, problems in higher dimensions can be investigated successfully. The combination with other codes promises interesting perspectives.

Acknowledgments

This work was supported by OTKA grant T 031744, FKFP grant 0177/2001 and the János Bolyai Research Fellowship (GD). The comments of Zsolt Gáspár, Réka Tóth and our anonymous refere are highly appreciated.

Appendix: Review of previously computed problems

Although the hybrid algorithm has only been tested on one 'real' problem ([6]), there is plenty of experience with the PSA and the SCA, which we summarize below.

- In **2D** an eaarlier version of the PSA was applied to compute the global bifurcation diagram of discrete elastic chains in [13] and [7]. The equilibria of continuous non-uniform elastic beams are determined in [8]. [12] solves the global bifurcation diagrams for liquid bridges with the current version of the PSA; the evaluated function is discontinuous.
- In **3D** the equilibria of symmetric masonry arches are computed via SCA in [21], due to material nonlinearity the evaluated functions are non-smooth and discontinuous. An earlier PSA version was applied in [3] to find the global bifurcation diagram of clamped-clamped and simply supported planar elastic beams. The stability diagrams related to [12] are computed in [11] via the recursive PSA. In [23] the PSA is used to explore the vicinity of a degenerate bifurcation point.
- In **4D** global search is performed by the PSA in [5] to find the spatial equilibria of twisted rings. Related to the same problem, in [16] the authors use the PSA to find equilibria in self-contact, here and in [22] the SCA is also applied. The equilibria of planar rings is investigated in [24].
- In **5D** large-scale global search was done by the PSA in [9] and [20] supplemented by continuation with the SCA to identify equilibria of planar elastic rods constrained between rigid parallel walls.
- In **6D** the first real application of the PHA is described in [6] where the equilibria of filaments with initial curvature are described. This is the highest dimension in which scanning was performed until now. In this example the ratio of the PHA's and the PSA's speed was over 100, cf. also Subsec. 6.3. In [20] the SCA is used in this dimension as well.
- in **7D** the equilibrium path of a two-storey planar frame is computed via SCA in [4]. In [20] the SCA is used in this dimension as well.
- in **8D** the equilibria of planar elasto-plastic frames are computed via SCA in [15]. The dimension of the GRS changes as the path is followed due to the appearence of plastic hinges, the maximal dimension is 10. In [20] the SCA is used in 8D and 9D as well.

The above listed references are summarized in Table 18 according to the applied method and the dimension of the GRS.

Table 18. Overview of applications

DIM:	2	3	4	5	6	7	8	9	10
PSA	[12], [13], [7], [8]	[3], [19], [11]	[5], [16], [23]	[20]					
SCA		[20], [21]	[20], [22]	[9], [20]	[20]	[4], [20]	[15], [20]	[15], [20]	[15]
PHA					[6]				

References

[1] E. L. Allgower, K. Georg. Numerical continuation methods: an introduction. *Springer-Verlag*, Berlin, 1990.

[2] E. Doedel, H. B. Keller, J. P. Kernevez. Numerical analysis and control of bifurcation problems (I) bifurcation in finite dimensions. *Int. J. Bifurcation and Chaos*, **1**(3:) 493–520, 1991.

[3] G. Domokos. Global description of elastic bars. *Zeitschr. Angew. Math. und Mech.*, **74**(4): T289–T291, 1994.

[4] G. Domokos, Zs. Gáspár. A global, direct algorithm for path-following and active static control of elastic bar structures. *Int'l J. of Structures and Machines*, **23**(4): 549–571, 1995.

[5] G. Domokos, T. Healey. Hidden symmetry of global solutions in twisted elastic rings. *J. Nonlinear Sci.*, **11**: 47–67, 2001.
[6] G. Domokos, T. Healey. Global description of intristically curved rods. manuscript, in preparation, 2002
[7] G. Domokos, P. Holmes. Euler's problem and Euler's method, or the discrete charm of buckling. *J. Nonlin. Sci.*, **3:** 109–151, 1993.
[8] G. Domokos, P. Holmes. On non-inflexional solutions of non-uniform elasticae. *Int. J. Nonlin. Mech.*, **28:**(6) 677–685, 1993.
[9] G. Domokos, P. Holmes, B. S. H. Royce. Constrained Euler buckling. *J. Nonlin. Sci.*, **7**: 1–34, 1997.
[10] G. Domokos, R. Paffenroth. A case study on visualization tool for boundary value problems. In IEEE Computer Society Visualization '94 Conference Proceedings *IEEE Comp. Soc.*, 345–350, Press, 1995.
[11] G. Domokos, I. Szeberényi, P. Steen. Parallel, recursive computation of global stability charts for liquid bridges. In Proc. 6th EuroPVM/MPI2000, *Springer (Lec. Notes in Comp. Sci.*, **1908**: 64–71, 2000.
[12] G. Domokos, I. Szeberényi, P. Steen. Simulatenously resolved bifurcation diagrams: a novel global approach to liquid figures of equilibrium. *J. Comp. Phys.*, **159**: 38–57, 2000.
[13] Z. Gáspár, G. Domokos. Global investigation of discrete models of the Euler buckling problem. *Acta Techn. Hung.*, **102**: 227–238, 1989.
[14] Z. Gáspár, G. Domokos, I. Szeberényi. A parallel algorithm for the global computation of elastic bar structures. *Comp. Assist. Mech. and Eng. Sci.*, **4:** 55–68, 1997.
[15] Z. Gáspár P. Nédli. Global numerical analysis of elasto-plastic frames. *Comp.Ass. Mech. and Eng. Sci.*, **5**: 93–100, 1998.
[16] Z. Gáspár R. Németh. A special shape of a twisted ring. In *Proc. 2nd European Conference on Computational Mechanics*, 11, Compact Disc, Cracow University, 2001.
[17] A. Geist and al. Pvm 3 user's guide and reference manual. *Oak Ridge National Laboratory, Technical Report ORNL/TM-12187*, 1993.
[18] J. Guckenheimer, M. R. Myers, F. J. Wicklin, P. A. Worfolk. dstool: A dynamical system toolkit with an interactive graphical interface. *Center for Applied Mathematics, Cornell University*, 1991.
[19] P. Holmes, G. Domokos, G. Hek. Euler buckling in a potential field. *J. Nonlinear Science*, **10**: 477–505, 2000.
[20] P. Holmes, G. Domokos, J. Schmitt, and I. Szeberényi. Constrained Euler buckling: an interplay of computation and analysis. *Comp. Meth. in Appl. Mech. and Eng.*, **170**(3-4): 175–207, 1999.
[21] R. Macskási. Investigation of masonry arches. *Proc. Estonian Acad. Sci. Math. and Phys.*, in press.
[22] R. Németh. Determining the shape of twisted rings. In *Proc. 3rd Int. PhD Symposium*, 319–326, T.U Wien, 2000.
[23] S. Pajunen, Z. Gáspár. Study of an interactive buckling model – from local to global approach. In J. Rondal, D. Dubina, V. Gioncu, editors, *Proc. 2nd Int. Conf. on Coupled instabilities in metal structures*, 35–42, Imperial College Press, London, 1996.
[24] S. Pajunen, M. Toumala. Nonlinear stability analysis of statically loaded structures. *Technical Report, Univ. Tampere*, **1**: 21, 1997.
[25] E. Riks. An incremental approach to the solution of snapping and buckling problems. *Int. J. Solids Structures*, **15**: 529–551,1979 .