Parallel Global Optimization with the Particle Swarm Algorithm

J.F. Schutte[1], J.A. Reinbolt[2], B.J. Fregly[1,2], R.T. Haftka[1], A.D. George[3]

[1] *Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL*

[2] *Department of Biomedical Engineering, University of Florida, Gainesville, FL*

[3] *Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL*

-

Address correspondence to:

B.J. Fregly, Ph.D.

Department of Mechanical & Aerospace Engineering

231 MAE-A Building

P.O. Box 116250

University of Florida

Gainesville, FL 32611-6250

Email: fregly@ufl.edu

Phone: (352) 392-8157

Fax: (352) 392-7303

SUMMARY

Present day large-scale engineering optimization problems impose large computational demands, resulting in long solution times even on modern high-end processors. To obtain enhanced computational throughput and global search capability, we detail the parallelization of an increasingly popular global search method, the Particle Swarm Optimization (PSO) algorithm. The parallel PSO algorithm's robustness and efficiency are demonstrated using a biomechanical system identification problem containing several local minima and numerical or experimental noise. The problem involves finding the kinematic structure of an ankle joint model that best matches experimental movement data. For synthetic movement data generated from realistic ankle parameters, the algorithm correctly recovered the known parameters and produced identical results to a synchronous serial implementation of the algorithm. When numerical noise was added to the synthetic data, the algorithm found parameters that reduced the fitness value below that of the original parameters. When applied to experimental movement data, the algorithm found parameters consistent with previous investigations and demonstrated an almost linear increase in throughput for up to 30 nodes in a computational cluster. Parallel PSO provides a new option for global optimization of large-scale engineering problems.

KEY WORDS: Particle swarm, parallel optimization, biomechanical model, cluster computing.

INTRODUCTION

Numerical optimization has been widely used in engineering to solve a variety of NP-complete problems in areas such as structural optimization, neural network training, control system analysis and design, and layout and scheduling problems to name but a few. In these and other engineering disciplines, two major obstacles limiting the solution efficiency are frequently encountered. First,

large-scale problems are often computationally demanding, requiring significant resources in time and hardware to solve. Second, engineering optimization problems are often plagued by multiple local optima and numerical noise, requiring the use of global search methods such as population-based algorithms to deliver reliable results.

Fortunately, recent advances in microprocessor technology and network technology have led to increased availability of low cost computational power through clusters of low to medium performance computers. To take advantage of this, communication layers such as MPI and PVM have been used to develop parallel optimization algorithms, the most popular being gradient-based, genetic (GA), and simulated annealing (SA) algorithms [1, 2, 3]. In biomechanical optimizations of human movement, for example, parallelization has allowed previously intractable problems to be solved in a matter of hours [1].

The Particle Swarm Optimization (PSO) algorithm is a recent addition to the list of global search methods [4]. This derivative free method is particularly suited to continuous variable problems and has received increasing attention in the optimization community. It has been successfully applied to large-scale problems [5, 6, 7] in several engineering disciplines and, being a population based approach, is readily parallelizable. It also has fewer algorithm parameters than either GA or SA algorithms. Furthermore, generic settings for these parameters work well on most problems [8, 9].

In this study study, we present a parallel PSO algorithm for application to large-scale problems. The algorithm's robustness to local minima and enhanced throughput due to parallelization are demonstrated on a sample biomechanical system identification problem. The problem involves fitting a parametric ankle joint kinematic model to synthetic and experimental movement data defined by trajectories of surface markers [10].

SERIAL PARTICLE SWARM ALGORITHM

Particle swarm optimization was introduced in 1995 by Kennedy and Eberhart [11]. Although several modifications to the original swarm algorithm have been made to improve performance [12, 13, 14, 15, 16] and adapt it to specific types of problems [6, 17, 18], a parallel version has not been previously implemented.

The following is a brief introduction to the operation of the particle swarm algorithm. Consider a flock or swarm of p particles, with each particle's position representing a possible solution point in the design problem space D. For each particle i, Kennedy and Eberhart proposed that the position $\boldsymbol{x}^i$ be updated in the following manner:

$$\boldsymbol{x}^i_{k+1} = \boldsymbol{x}^i_k + \boldsymbol{v}^i_{k+1}, \tag{1}$$

with a pseudo-velocity $\boldsymbol{v}^i_{k+1}$ calculated as follows:

$$\boldsymbol{v}^i_{k+1} = w_k \boldsymbol{v}^i_k + c_1 r_1 (\boldsymbol{p}^i_k - \boldsymbol{x}^i_k) + c_2 r_2 (\boldsymbol{p}^g_k - \boldsymbol{x}^i_k). \tag{2}$$

Here, subscript k indicates a (unit) pseudo-time increment, $\boldsymbol{p}^i_k$ represents the best ever position of particle i at time k (the cognitive contribution to the search vector $\boldsymbol{v}^i_{k+1}$), and $\boldsymbol{p}^g_k$ represents the global best position in the swarm at time k (social contribution). r_1 and r_2 represent uniform random numbers between 0 and 1. To allow the product $c_1 r_1$ or $c_2 r_2$ to have a mean of 1, Kennedy and Eberhart proposed that the cognitive and social scaling parameters c_1 and c_2 be selected such that $c_1 = c_2 = 2$. The result of using these proposed values is that the particles overshoot the target half the time, thereby maintaining separation within the group and allowing for a greater area to be searched. The addition of the variable w_k, set at 1 at initialization, is a modification to the original PSO algorithm [13]. This allows a more refined search as the optimization progresses by reducing its value linearly or dynamically [6].

The serial PSO algorithm as it would typically be implemented on a single CPU computer is described below, where p is the total number of particles in the swarm. The best ever fitness value of a particle at design coordinates $\boldsymbol{p}_k^i$ is denoted by f_{best}^i and the best ever fitness value of the overall swarm at coordinates $\boldsymbol{p}_k^g$ by f_{best}^g. At the initialization timestep $k = 0$, the particle velocities $\boldsymbol{v}_0^i$ are initialized to random values within the limits $0 \leq \boldsymbol{v}_0 \leq \boldsymbol{v}_0^{max}$. The vector $\boldsymbol{v}_0^{max}$ is calculated as a fraction of the distance between the upper and lower bounds $\boldsymbol{v}_0^{max} = \zeta(\boldsymbol{x}_{UB} - \boldsymbol{x}_{LB})$ [6], with $\zeta = 0.5$.

1. Initialize

   (a) Set constants k_{max}, c_1, c_2, w_0

   (b) Randomly initialize particle positions $\boldsymbol{x}_0^i \in \boldsymbol{D}$ in $\mathbb{R}^n$ for $i = 1, ..., p$

   (c) Randomly initialize particle velocities $0 \leq \boldsymbol{v}_0^i \leq \boldsymbol{v}_0^{max}$ for $i = 1, ..., p$

   (d) Set $k = 1$

2. Optimize

   (a) Evaluate function value f_k^i using design space coordinates $\boldsymbol{x}_k^i$

   (b) If $f_k^i \leq f_{best}^i$ then $f_{best}^i = f_k^i$, $\boldsymbol{p}^i = \boldsymbol{x}_k^i$

   (c) If $f_k^i \leq f_{best}^g$ then $f_{best}^g = f_k^i$, $\boldsymbol{p}^g = \boldsymbol{x}_k^i$

   (d) If stopping condition is satisfied then go to 3.

   (e) Update particle velocity vector $\boldsymbol{v}_{k+1}^i$ using Eq. (2).

   (f) Update particle position vector $\boldsymbol{x}_{k+1}^i$ using Eq. (1).

   (g) Increment i. If $i > p$ then increment k, and set $i = 1$.

   (h) Go to 2(a).

3. Report results

4. Terminate

The above is illustrated as a flow diagram in Figure 1

PARALLEL PARTICLE SWARM ALGORITHM

The following issues had to be taken into consideration to create a parallel PSO algorithm.

Concurrent Operation and Scalability

The algorithm should operate in such a fashion that it can be easily decomposed for parallel operation on a multi-processor machine. Furthermore, it is highly desireable that it be scalable. This implies that the nature of the algorithm should not place a limit on the amount of computational nodes that can be utilized.

An example of an algorithm with limited scalability is a parallel implementation of a gradient-based algorithm. This algorithm is decomposed by distributing the workload of the derivative calculations for a single point in design space among multiple processors. The upper limit on concurrent operations using this approach is therefore set by the number of variables in the problem.

On the other hand, population-based methods such as the GA and PSO algorithms are better suited to parallel computing. Here the population of individuals representing designs can be increased or decreased according to the availability and speed of processors. Any additional agents in the population will allow for a higher fidelity search in the design space, lowering susceptibility to entrapment in local minima. However, this comes at the expense of additional fitness evaluations.

Asynchronous vs. Synchronous Implementation

The original PSO algorithm was implemented with a synchronized scheme for updating the best "remembered" individual and group fitness values, f_k^i and f_k^g, and their associated positions $\boldsymbol{p}_k^i$ and $\boldsymbol{p}_k^g$. This entails performing the fitness evaluations for the entire swarm before updating the best fitness values. Subsequent experimentation revealed that improved convergence rates can be obtained by updating the f_k^i and f_k^g values and their positions after each individual fitness evaluation (i.e., in an asynchronous fashion) [8, 9]. It is speculated that because the updating occurs immediately after each fitness evaluation, the swarm reacts more quickly to an improvement in the best-found fitness value.

With the parallel implementation, however, this asynchronous improvement on the swarm is lost since fitness evaluations are performed concurrently. The parallel algorithm requires updating f_k^i and f_k^g for the entire swarm after all fitness evaluations have been performed, as in the original particle swarm formulation. Consequently, the swarm will react more slowly to changes of the best fitness value "position" in the design space. This produces an unavoidable performance loss in terms of convergence rate compared to the asynchronous implementation and can be considered part of the overhead associated with parallelization.

Coherence

Parallelization should have no adverse affect on algorithm operation. Calculations sensitive to program order should appear to have occurred in exactly the same order as in the original formulation, leading to the exact same final answer as obtained by a serial implementation. In the serial PSO algorithm the fitness evaluations form the bulk of the computational effort for the optimization and can be performed independently. For our parallel implementation, we therefore chose to decompose the algorithm to perform the fitness evaluations concurrently on a parallel machine. Step 2 of the particle swarm

optimization algorithm was then modified accordingly to operate in a parallel manner:

2. Optimize

   (a) Evaluate all i particle fitness values f_k^i in parallel using design space coordinates $\boldsymbol{x}_k^i$.

   (b) Perform barrier synchronization of all fitness evaluation results.

   (c) If $f_k^i \leq f_{best}^i$ then $f_{best}^i = f_k^i$, $\boldsymbol{p}_k^i = \boldsymbol{x}_k^i$.

   (d) If $f_k^i \leq f_{best}^g$ then $f_{best}^g = f_k^i$, $\boldsymbol{p}_k^g = \boldsymbol{x}_k^i$.

   (e) If stopping condition is satisfied then go to 3.

   (f) Update all particle velocities $\boldsymbol{v}_k^i$ for $i = 1, ..., p$ with Eq. (2).

   (g) Update all particle positions $\boldsymbol{x}_k^i$ for $i = 1, ..., p$ with Eq. (1).

   (h) Increment k.

   (i) Go to 2(a).

The parallel PSO algorithm is represented by the flow diagram in Figure 2

Network Communication

In a parallel computational environment, the main performance bottleneck is the communication latency between processors. This is especially true for large clusters of computers where the use of high performance network interfaces are limited due to their high cost. To keep communication between different computational nodes at a minimum, fitness evaluation tasks are used as the level of granularity for the parallel software. As previously mentioned, each of these evaluations can be performed independently and requires no communication aside from receiving design space coordinates to be evaluated and reporting the fitness value at the end of the analysis.

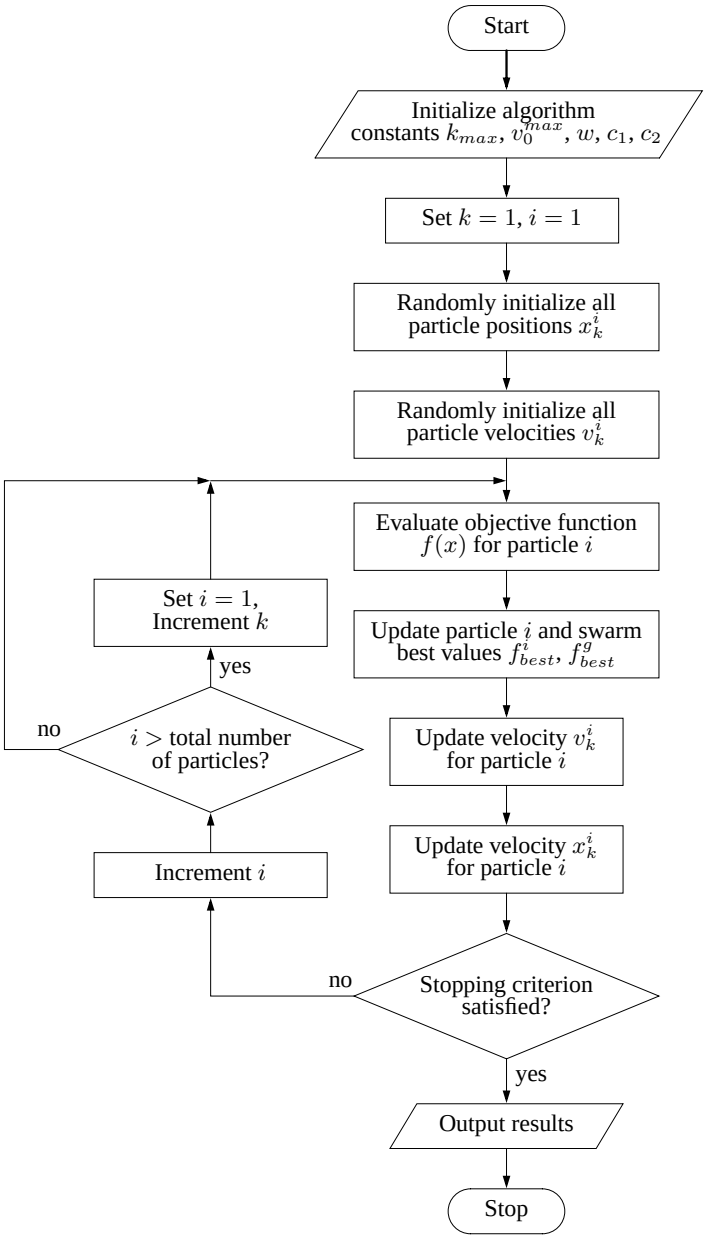


Figure 1. Serial implementation of PSO algorithm

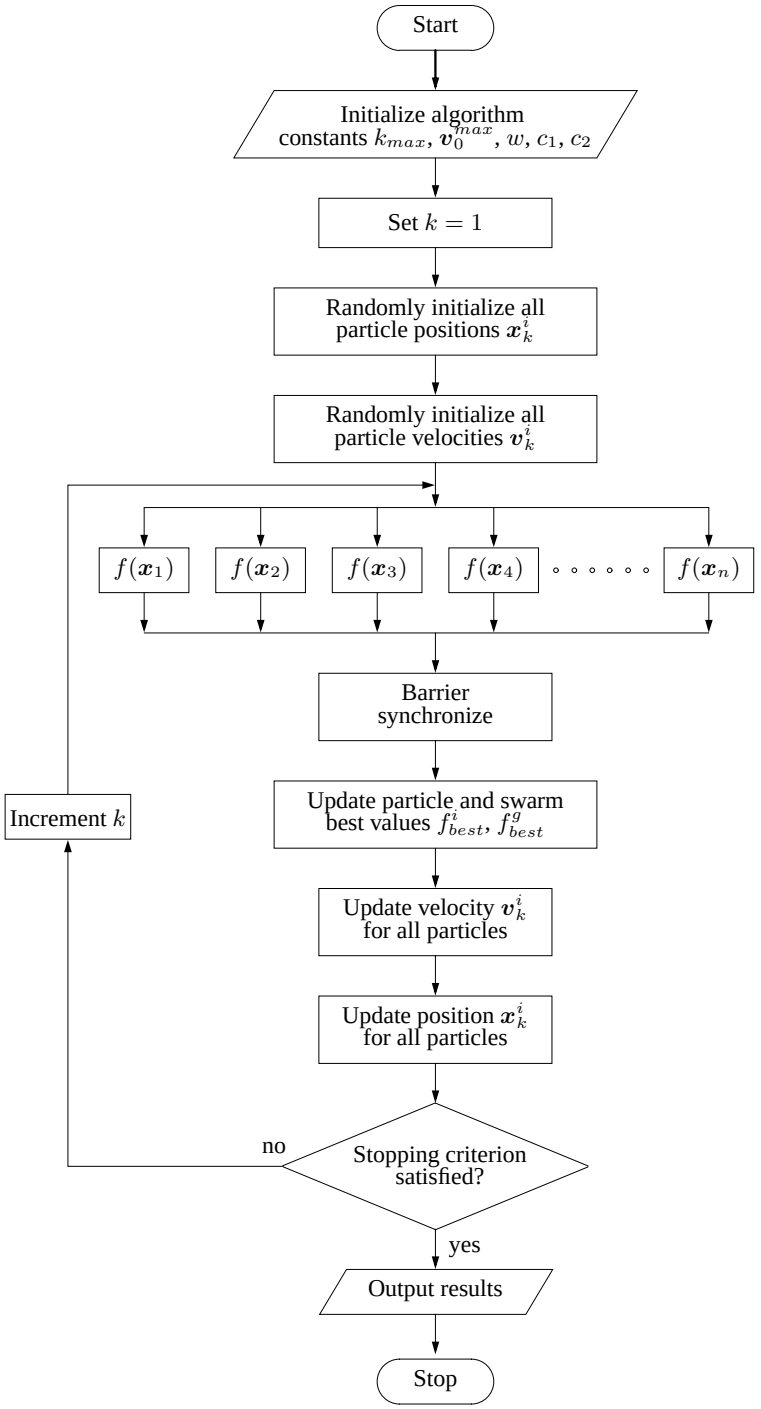


Figure 2. Parallel implementation of PSO algorithm

The optimization infrastructure is organized into a coordinating node and several computational nodes. PSO algorithm functions and task orchestration are performed by the coordinating node, which assigns the design coordinates to be evaluated, in parallel, to the computational nodes. With this approach, no communication is required between computational nodes as individual design fitness evaluations are independent of each other. The only necessary communication is between the coordinating node and the computational nodes and encompasses the following:

1. Several distinct design variable configuration vectors assigned by coordinating node to slave nodes for fitness evaluation.
2. Fitness values reported from slave nodes to coordinating node.
3. Synchronization signals to maintain program coherence.
4. Termination signals from coordinating node to slave nodes on completion of analysis in order for the program to stop cleanly.

Synchronization

From the parallel implementation algorithm, it is clear that some means of synchronization is required to ensure that all of the particle fitness evaluations have been completed and results reported before the velocity and position calculations can be executed (steps 2c and 2d). This is done by using a global synchronization or barrier function in the MPI communication library which temporarily stops the coordinating node from proceeding with the next swarm iteration until all of the computational nodes have responded with a fitness value. This, however, implies that the time required for a single parallel swarm fitness evaluation will be dictated by the slowest fitness evaluation in the swarm.

Implementation

The parallel PSO scheme and the required communication layer was implemented in ANSI C on the Linux operating system with the message passing interface (MPI) libraries [19]. A cluster of 40 1.3 GHz Athlon personal computers was used to obtain the numerical results.

SAMPLE BIOMECHANICAL APPLICATION

For our example problem, we chose a system identification problem of a biomechanical nature. This computationally-intensive large-scale optimization attempts to determine the kinematic structure of a parametric ankle joint model from experimental surface marker data. The experimental data is gathered by using an optoelectronic system. The system uses multiple cameras to record the positions of external markers placed at certain locations on the body segments. To permit measurement of three-dimensional motion, three non-colinear markers are attached to the foot and lower leg. The recordings are processed to obtain marker trajectories in a laboratory fixed coordinate system [20], [21].

The first step in the system identification procedure is to formulate a parametric ankle joint model that will emulate the patient's movement by having sufficient degrees of freedom. The complexity of the parametric model depends on the purpose of the end application. Any added complexity will translate into increased computational effort at simulation time. Consequently, some tradeoff must be made, taking into consideration the available computational power and time that can be committed to solving the problem.

For the purpose of this paper, we approximate the talocrural and subtalar joints as simple 1 degree of freedom revolute joints. As shown in Figure 3, the selected ankle joint model contains 12 adjustable parameters that define its kinematic structure [10]. Each parameter is a joint axis position or orientation

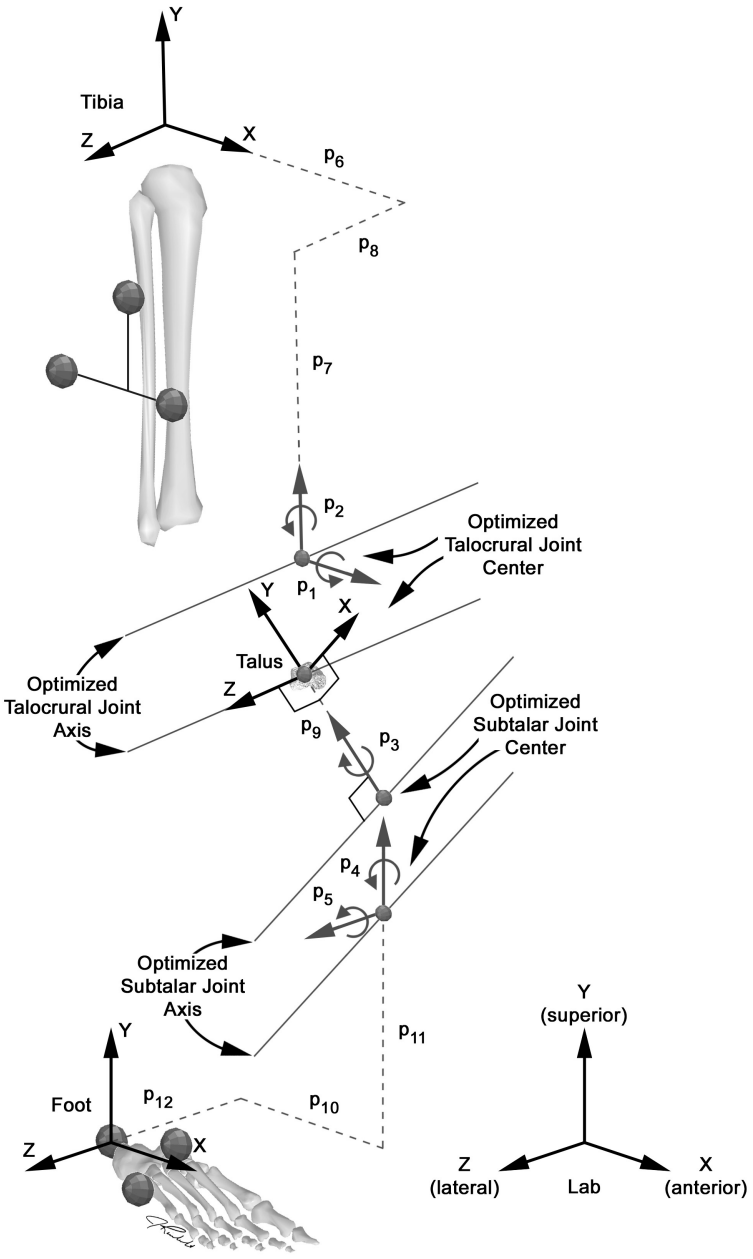


Figure 3. Joint locations and orientations in parametric kinematic ankle model. Each p_i (i = 1,...,12) represents a different position or orientation parameter in the model.

in the lower leg, talus, or foot segment. This kinematic model also has a set of virtual markers fixed to the limb segments in positions corresponding to those of the measured subject. The linkage parameters are then adjusted via optimization until the model is able to accurately emulate the rigid body motions dictated by the marker trajectory data.

To quantify how closely the kinematic model can follow experimental marker trajectories, we define a cumulative marker error e as follows:

$$e = \sum_{j=1}^{n} \sum_{i=1}^{m} {\Delta_{i,j}}^2 \tag{3}$$

with

$$\Delta_{i,j}^2 = \Delta x_{i,j}^2 + \Delta y_{i,j}^2 + \Delta z_{i,j}^2. \tag{4}$$

where $\Delta x_{i,j}$, $\Delta y_{i,j}$ and $\Delta z_{i,j}$ are the spatial displacement errors in the x, y, and z directions as measured in the laboratory coordinate system, for marker i, in frame j. These errors are calculated between the experimental marker locations on the human subject and the virtual marker locations on the kinematic model. The cumulative error becomes the objective function to be minimized. For each time frame, a nonlinear least squares sub-optimization is performed to determine the joint angles that minimize $\Delta_{i,j}^2$ for the current set of model parameters. By varying the model parameters, the PSO algorithm finds the best model that minimizes e over all time frames.

The experimental marker trajectory data consist of $n = 50$ time frames for a total of 111 experimental marker coordinates (i.e., 37 markers) on the subject. Only the relevant subset of m = 6 markers, namely 3 lower leg and 3 foot markers, are of interest for the kinematic ankle model. These marker trajectories define the motion of the foot and lower leg, both of which are assumed to be rigid bodies, in the laboratory reference frame.

For the numerical testing, three data sets were analyzed as described below, where the number of particles used in the swarm for each optimization was 20.:

1. *Synthetic data without numerical noise*

   Synthetic data without numerical noise were generated by simulating marker movements using a lower body kinematic model with virtual markers. The synthetic motion was based on an experimentally measured ankle motion (see 3 below). The kinematic model used anatomically realistic joint positions and orientations. Since the joint parameters associated with the synthetic data were known, this optimization was used to verify that the parallel PSO algorithm could accurately recover the original model.

2. *Synthetic data with numerical noise*

   Stochastic numerical noise was superimposed on each synthetic marker coordinate trajectory to emulate the effect of marker displacements caused by skin movement artifacts [22]. A previously published noise model requiring three random parameters was used to generate a marker perturbation N [23]:

$$N = Asin(\omega t + \phi) \tag{5}$$

   where A is the amplitude, ω the frequency, and ϕ the phase angle of the noise. The random variables were scaled to fall within the bounds $0 \leq A \leq 1\ cm$, $0 \leq \omega \leq 25\ rad/sec$, and $0 \leq \phi \leq 2\pi$ [23].

3. *Experimental data*

   Experimental marker trajectory data were obtained by processing three-dimensional recordings of a subject performing movements with reflective markers attached to the foot and shank as previously described. Institutional review board approval was obtained for the experiments and data analysis, and the subject gave informed consent prior to participation. Marker positions were reported in a laboratory fixed coordinate system.

NUMERICAL RESULTS

Synthetic and experimental optimization results are summarized in Tables I and II. In the tabulated results, p_1 to p_{12} are the kinematic model parameters and N_{fe} is the number of fitness evaluations (sub-optimizations of 50 timeframes) per analysis.

	Upper bound	Lower bound	Synthetic solution	Synthetic data without noise	Synthetic data with noise
N_{fe}				40000	40000
p_1 (degrees)	48.66935	-11.633065	18.366935	18.364964	15.13010
p_2 (degrees)	30.0	-30.0	0.0	-0.011809	8.00750
p_3 (degrees)	70.230969	10.230969	40.230969	40.259663	32.97410
p_4 (degrees)	53.0	-7.0	23.0	23.027088	23.12202
p_5 (degrees)	72.0	12.0	42.0	42.002080	42.03973
p_6 (cm)	6.270881	-6.270881	0.0	0.000270	-0.39360
p_7 (cm)	-33.702321	-46.244082	-39.973202	-39.972852	-39.61422
p_8 (cm)	6.270880	-6.270881	0.0	-0.000287	0.75513
p_9 (cm)	0.0	-6.270881	-1.0	-1.000741	-2.81694
p_{10} (cm)	15.266215	2.724454	8.995334	8.995874	10.21054
p_{11} (cm)	10.418424	-2.123338	4.147543	4.147353	3.03367
p_{12} (cm)	6.888097	-5.653664	0.617217	0.616947	-0.19037

Table I. Optimization results for synthetic marker trajectories without and with noise.

RMS Errors	Synthetic Data Without Noise	Synthetic Data With Noise	Experimental Data
Marker distances (cm)	3.58e-4	0.551	0.394
Orientation parameters (deg)	1.85e-2	5.05	N/A
Position parameters (cm)	4.95e-4	1.04	N/A

Table II. Synthetic and experimental marker distance and joint parameter RMS errors

As can be seen from Table I, the algorithm had no difficulty recovering the original parameters from the synthetic dateset without noise, with a final cumulative error value e on the order of 10^{-13}. This results in the optimum model being recovered with mean orientation errors less than 0.05 degrees and mean position errors less than 0.008 cm. Furthermore, the parallel implementation produced identical fitness and parameter histories as a synchronous serial implementation. For the synthetic data with superimposed noise, a RMS marker distance error of 0.551 cm was found, which is on the order of the imposed numerical noise with a maximum amplitude of 1 cm. This error corresponds to a RMS orientation error of 5.05 degrees and a mean position error of 1.04 cm. For the experimental data, the RMS marker distance error was 0.394 cm, which is on the same order of magnitude as for the synthetic data with noise. The recovered parameters were consistent with values in the literature [10].

Figures 4, 5, and 6 indicate convergence data for the three sets of data under consideration. As can be seen from the plots, the initial convergence rate is quite high, whereafter it slows when the approximate location of the minimum is found. After approximately 750 swarm iterations, there is little or no improvement in the fitness value and the algorithm can be considered to have converged.

DISCUSSION

The poor agreement of orientation parameters ($p_1 - p_4$) to actual values for the noisy synthetic dataset (Table I) are the result of the induced numerical noise. This can be explained by the dependency of orientation calculations on marker positions. Because of the close proximity of the markers to each other, even relatively small amplitude numerical noise in marker positions can result in large fluctuations in the best-fit joint orientations. This can be clearly seen in the RMS orientation error (Table II). To counter this, a larger dataset could be used to offset the effects of noise. Nonetheless the fitness value for the optimized parameters was lower than that obtained by evaluating the fitness using the original parameters.

Several local minima were observed when the noiseless synthetic data were analyzed with a gradient-based optimizer using random initial starting points. These minima are illustrated in Figure 7 by using an interpolating fitness plot on lines between three of local minima. To evaluate further the parallel PSO algorithm's ability to avoid entrapment in local minima, ten additional runs were performed using different initial random number seeds. In all ten cases, the algorithm converged to the global solution.

A scaling study was also undertaken to investigate the speedup obtained by solving a fixed amount of ankle parameter fitness evaluations (1000) with an increasing number of computational nodes from 5 to 30, with the swarm size set accordingly. The particle swarm has been proven to be relatively insensitive to the amount of particles in the swarm [8, 9], as long as the swarm size does not drop below a value of 5. Figure 8 shows analysis duration (wall clock times) per fitness evaluation as a function of number of nodes. As can be seen from this figure, using an increasing number of nodes leads to dramatically improved thoughput, obtaining near linear speedup rates. However, at the upper limit of this study, communication overhead began to degrade efficiency.

Large variations were observed for the time required to complete some of the concurrent processes

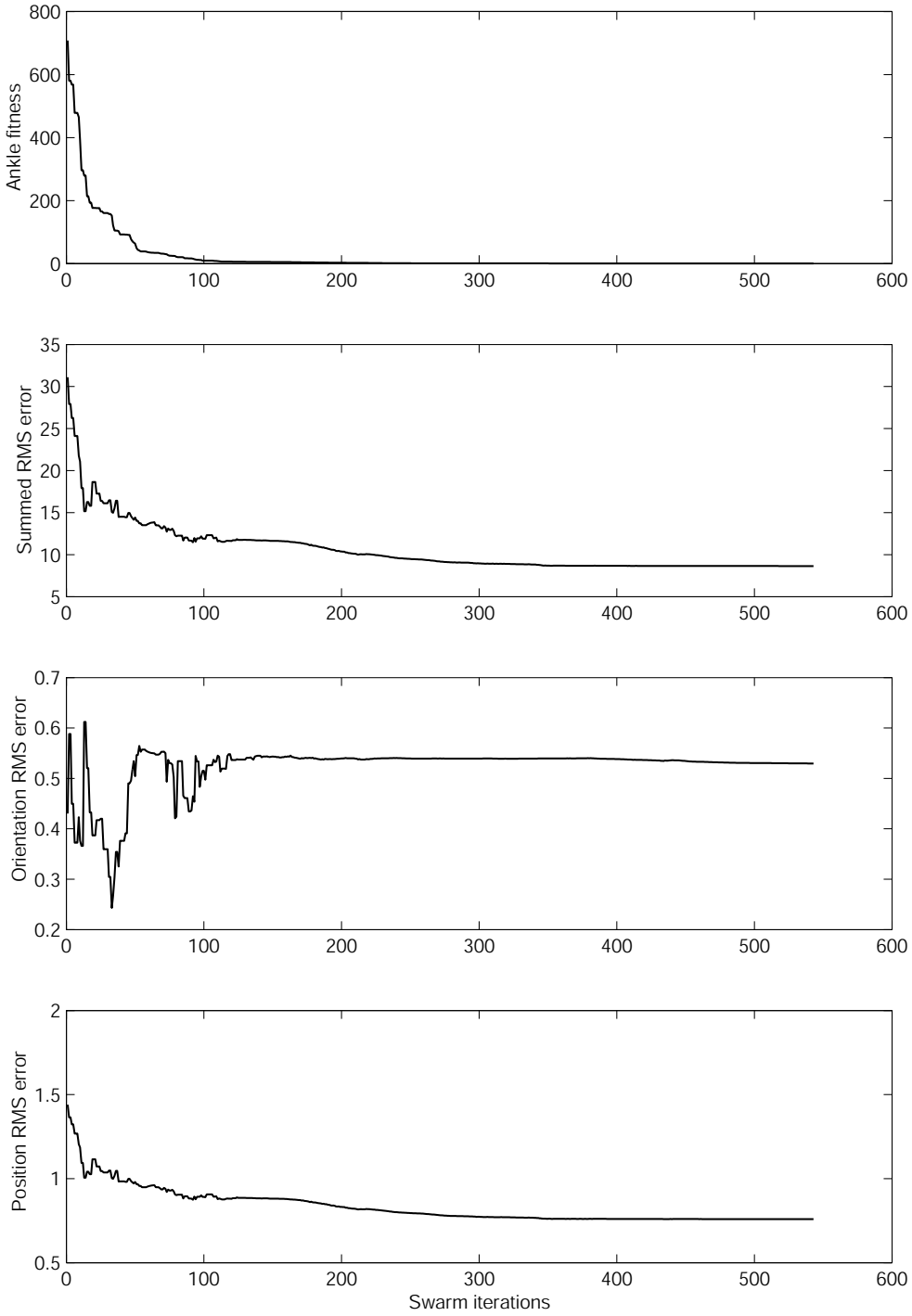


Figure 4. Fitness convergence and parameter error plots for synthetic ankle data without noise

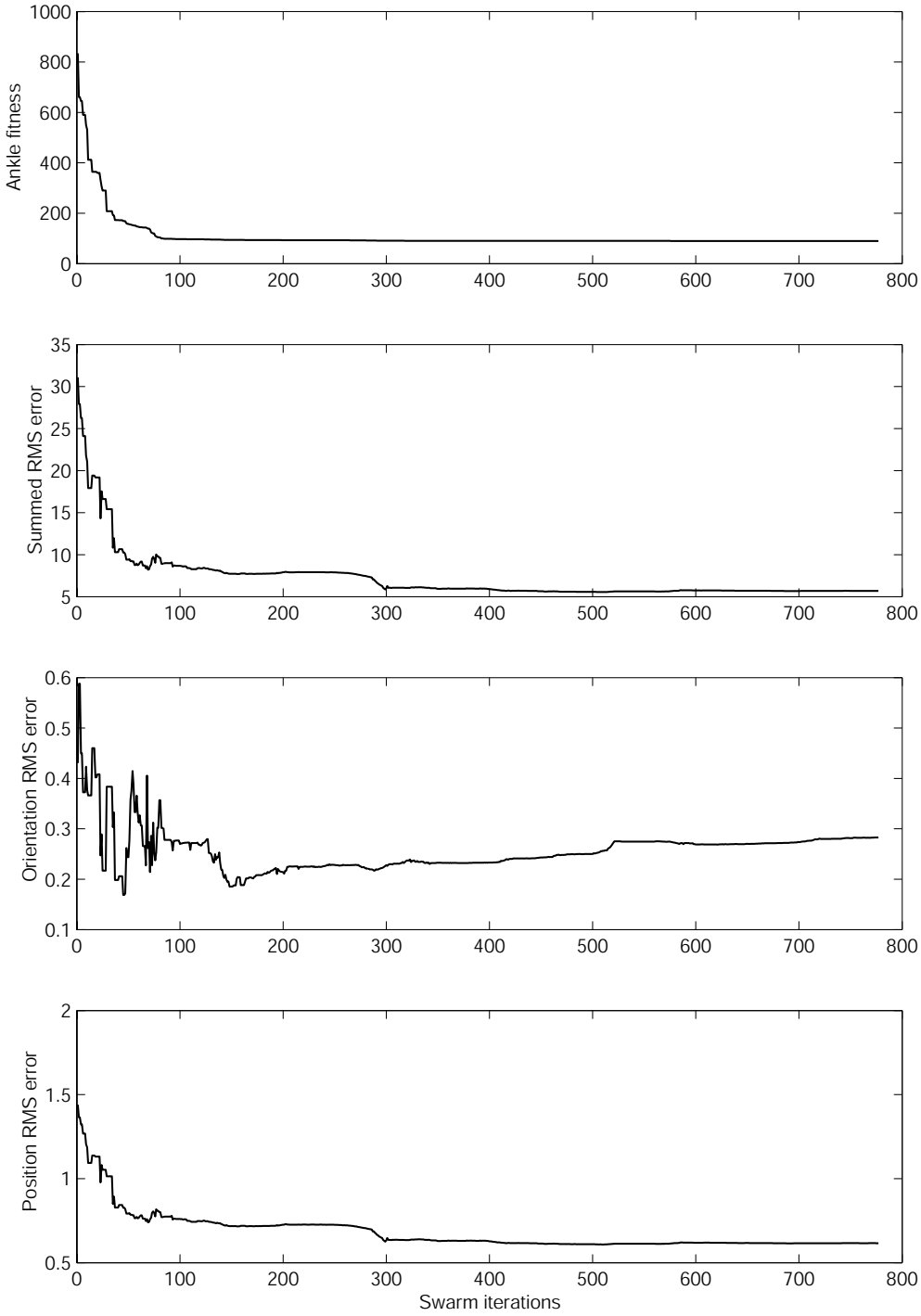


Figure 5. Fitness convergence and parameter error plots for synthetic ankle data with noiseplot

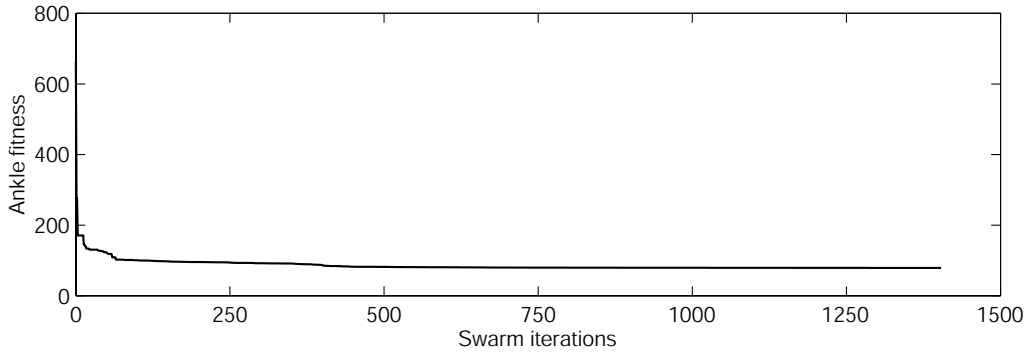
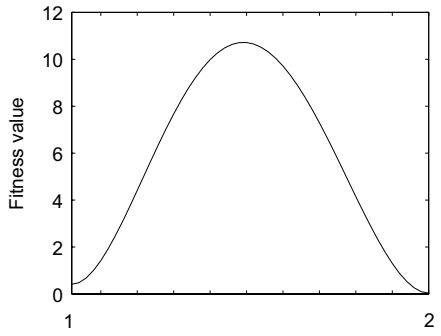


Figure 6. Fitness convergence plot for experimental data

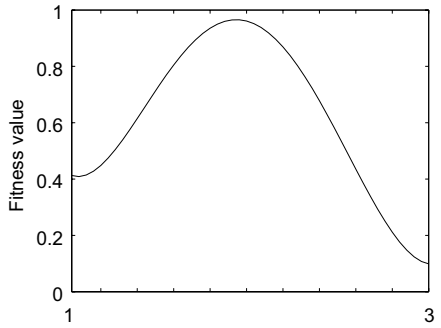
on different computational nodes. This can be explained in part by some fitness evaluations requiring more time to complete. Consequently, due to the synchronization requirement of the parallel PSO algorithm, the time taken by each parallel set of fitness evaluations on the cluster was dictated by the slowest concurrent process. This results in a single slow node severely degrading the entire analysis. An added drawback is that a loss of any node will result in the entire analysis being stopped.

CONCLUSIONS

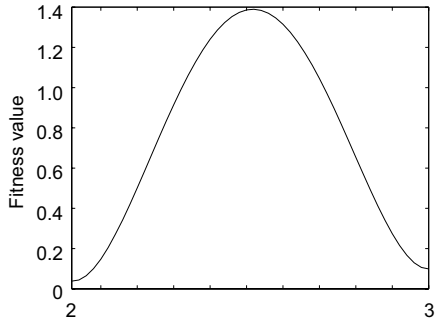
This study has presented a parallel implementation of the Particle Swarm Optimization algorithm. The method was validated by accurately recovering the parameters on a kinematic ankle model from synthetically generated data. Models were also created from synthetic data with numerical noise and experimental data collected from a human subject. By using parallel computation, the time required to solve the system identification problem was reduced substantially, proving that optimization using a PSO algorithm on a cluster of processors is a worthwhile option to solve large-scale optimization problems exhibiting multiple local minima. A possible avenue for future study will be investigating ways of modifying the PSO algorithm to obtain a more efficient and robust parallel scheme. This could by achieved by eliminating wasted CPU cycles via a dynamic task queue.



(a) Fitness value interpolation plot between solution 1 and 2



(b) Fitness value interpolation plot between solution 1 and 3



(c) Fitness value interpolation plot between solution 2 and 3

Figure 7. Presence of local minima

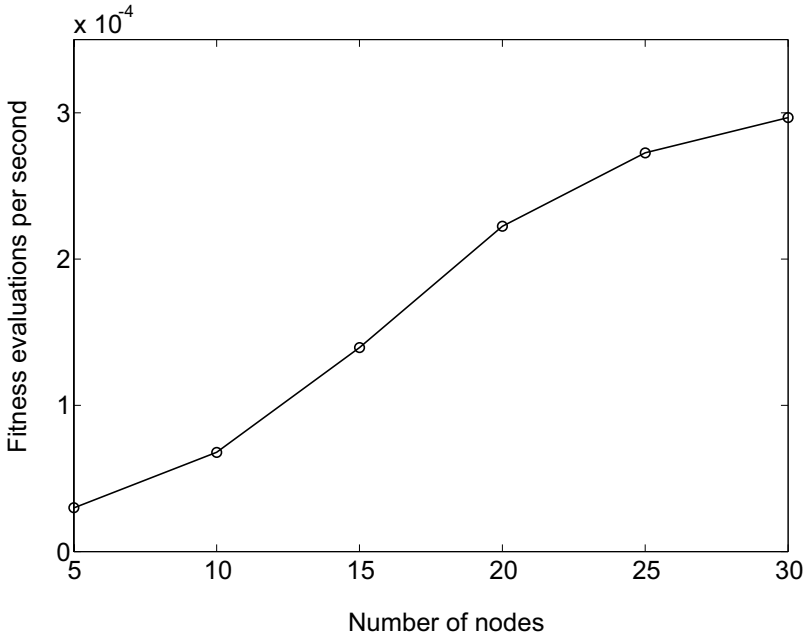


Figure 8. Speedup with increasing number of nodes

ACKNOWLEDGEMENTS

The authors gratefully acknowledge funding for this study from NIH National Library of Medicine (R03 LM07332-01) and Whitaker Foundation grants to B.J. Fregly and an AFOSR (F49620-09-1-0070) grant to R.T. Haftka.

REFERENCES

1. F.C. Anderson, J. Ziegler, M.G. Pandy, and R.T. Whalen. Application of of high-performance computing to numerical simulation of human movement. *Journal of Biomechanical Engineering* 1995;**117**:300–308.
2. A.J. van Soest and L.J.R. Casius. The merits of a parallel genetic algorithm in solving hard optimization problems. *Journal of Biomechanical Engineering* 2003; **125**:141–146.
3. B. Monien, F. Ramme, and H. Salmen. A parallel simulated annealing algorithm for generating 3D layouts of undirected

graphs. In Franz J. Brandenburg, editor, *Proceedings of the 3rd International Symposium of Graph Drawing*. Springer-Verlag: Berlin, 1995; 396–408.

4. Russell C. Eberhart and Yuhui Shi. Particle swarm optimization: Developments, applications, and resources. In *Proceedings of the 2001 Congress on Evolutionary Computation* 2001; 81–86.
5. G. Venter and J. Sobieszczanski-Sobieski. Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization. In *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization* 2002, Atlanta, GA.
6. P.C. Fourie and A.A. Groenwold. The particle swarm algorithm in topology optimization. In *Proceedings of the Fourth World Congress of Structural and Multidisciplinary Optimization* 2001; Dalian, China.
7. R. C. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. In *Proceedings of the IEEE Congress on Evolutionary Computation* 2001; 27–30.
8. J.F. Schutte. Particle swarms in sizing and global optimization. Master's thesis, University of Pretoria, Department of Mechanical Engineering, 2001.
9. A. Carlisle and G. Dozier. An off-the-shelf pso. In *Proceedings of the Workshop on Particle Swarm Optimization*, 2001; Indianapolis.
10. A.J. van den Bogert, G.D. Smith, and B.M. Nigg. In vivo determination of the anatomical axes of the ankle joint complex: an optimization approach. *Journal of Biomechanics* 1994; **12**:1477–1488.
11. J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks* 1995; **4**:1942–1948.
12. Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary computation* 1998; 69–73.
13. Yuhui Shi and Russel C. Eberhart. Parameter selection in particle swarm optimization. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII*. Springer:Berlin, 1998; 591–600.
14. R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 World Congress on Evolutionary Computation* 2000; 84–88.
15. Maurice Clerc. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation* 1999; **3**:1951–1957.
16. Morten Løvbjerg, Thomas Kiel Rasmussen, and Thiemo Krink. Hybrid particle swarm optimiser with breeding and subpopulations. In *Proceedings of the third Genetic and Evolutionary Computation Conference* 2001.
17. A. Carlisle and G. Dozier. Adapting particle swarm optimization to dynamic environments. In *International Conference on Artificial Intelligence* 2000; **1**:429–434.

18. J. Kennedy and R.C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the 1997 Conference on Systems, Man and Cybernetics* 1997; 4104–4109.
19. William Gropp and Ewing Lusk. *User's Guide for mpich, a Portable Implementation of MPI*. Argonne National Laboratory, Mathematics and Computer science division. http:// www.mcs.anl.gov/mpi/mpiuserguide/paper.html.
20. I. Soderkvist and P.A. Wedin. Determining the movements of the skeleton using well-configured markers. *Journal of Biomechanics* 1993; **26**:1473–1477.
21. C.W. Spoor and F.E. Veldpaus. Rigid body motion calculated from spatial co-ordinates of markers. *Journal of Biomechanics* 1980; **13**:391–393.
22. T.-W. Lu and J.J. O'Connor. Bone position estimation from skin marker co-ordinates using global optimization with joint constraints. *Journal of Biomechanics* 1999 **32**:129–134.
23. L. Cheze, B.J. Fregly, and J. Dimnet. A solidification procedure to facilitate kinematics analyses based on video system data. *Journal of Biomechanics* 1995; **28**:879–884.