

A Parallel Particle Swarm Optimizer

J.F. Schutte (1), B.J. Fregly (1), R.T. Haftka (1), A. D. George (2)

(1) Dept. of Mechanical & Aerospace Engineering
University of Florida
Gainesville, FL

(2) Dept. of Electrical & Computer Engineering
University of Florida
Gainesville, FL

1. Abstract

Time requirements for the solving of complex large-scale engineering problems can be substantially reduced by using parallel computation. Motivated by a computationally demanding biomechanical system identification problem, we introduce a parallel implementation of a stochastic population based global optimizer, the Particle Swarm Algorithm as a means of obtaining increased computational throughput. The Particle Swarm requires very few algorithmic parameters to define convergence behavior due to its simplicity, and, as a population based optimization method it is a natural candidate for concurrent computation. The parallelization of the Particle Swarm Optimization (PSO) algorithm is detailed and its performance and characteristics demonstrated for the biomechanical system identification problem as example.

2. Keywords: parallel computation, particle swarm optimization, biomechanical system identification.

3. Introduction

The method of Particle Swarm optimization was introduced in 1995 by Kennedy and Eberhart [1] and has been successfully applied to several different problems, including the training of neural networks, structural and topology optimization, and image recognition. The underlying principle of this stochastic population-based method is that of a number of agents coordinating their search patterns in the design space by communicating the locations of promising regions. Each agent i in a swarm of p particles, occupies a distinct point \mathbf{x}_k^i in the design space at time step k , and has a pseudo velocity \mathbf{v}_k^i and inertia w^i . At each iteration k of the optimization process particles evaluate their fitness f_k^i and adjust velocities according to the most promising location found by the swarm \mathbf{p}_k^g and themselves \mathbf{p}_k^i . Positions are updated as follows:

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i + \mathbf{v}_{k+1}^i \quad (1)$$

with velocity calculated by

$$\mathbf{v}_{k+1}^i = w^i \mathbf{v}_k^i + c_1 r_1 (\mathbf{p}_k^i - \mathbf{x}_k^i) + c_2 r_2 (\mathbf{p}_k^g - \mathbf{x}_k^i) \quad (2)$$

Regions of high fitness act as attraction basins to which the particles will converge and overshoot. c_1 and c_2 are cognitive and social weights assigned to individual best and the swarm best remembered positions in the design space, and are both set at values of 2 to allow for particles to overshoot half of the time. r_1 and r_2 are uniformly distributed random numbers between 0 and 1. The cognitive (individual) and social (swarm) contributions to an individual particle's search direction (2) can be visualized as a parallelogram in a 2-D search space (Figure 1). By reducing the particle inertia w^i and limiting velocities as the search progresses the amount of overshoot can be reduced, thereby forcing the search region to become localized. This derivative-free method is an attractive approach to global optimization because of the small amount of algorithmic parameters required for its operation, which are the maximum velocity at initialization v_0 , inertia w^i , the rates these are reduced at, and values of c_1 and c_2 (usually set at 2). For our implementation we use a dynamic inertia and maximum velocity reduction scheme [2] to obtain a progressively reduced search area.

Parallel optimization with similar global methods such as Genetic Algorithms (GA's) and Simulated Annealing (SA) have been successfully applied to the optimization of complex problems by numerous authors [3,4,5,6]. The PSO algorithm is particularly suited to continuous variable problems for which gradient information is not available or very expensive to calculate. Although several modifications to the original swarm algorithm have been made to improve performance and reliability [7], a parallel version has not previously been implemented. We applied the PSO to a biomechanical system identification problem, entailing the reconstruction of a kinematic skeletal model of a human ankle joint from non-invasive experimental measurements [8]. To deal with large computational demands of this problem we parallelized the algorithm to obtain increased throughput. This paper outlines our approach to obtain a parallel implementation of the PSO algorithm and describes its performance for the biomechanical system identification problem.

4. Algorithm parallelization

Our parallelization approach is as follows: In order to minimize inter-nodal communication, which often forms the performance bottleneck on networked machines, we require coarse grain task division. By taking advantage of the fact that each particle can function individually with a minimal amount of communication, we decompose the algorithm by assigning each particle fitness evaluation to run as a separate process. These processes are then distributed evenly between available CPU's. A master/slave communication model is used to assign fitness evaluations and maintain algorithm synchronization.

For our application the master node is used exclusively for the algorithm operation, and slave nodes perform fitness evaluations of design configurations received via the message passing interface protocol (MPI) implementation. Communication between nodes/particles operates in a lock step or synchronous fashion, with all information on fitness and particle positions being exchanged between master and slave nodes at the end of a swarm movement cycle.

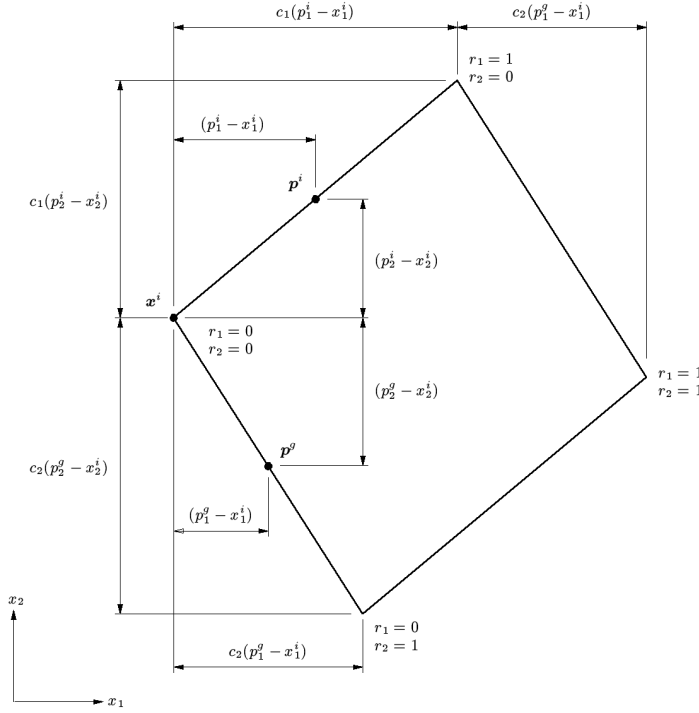


Figure 1. Cognitive and social search contributions.

The original serial implementation of the particle swarm can be outlined as follows:

1. Initialize
 - (a) Set constants $k_{max}, W^i, C_1, C_2, v_0^{max}$.
 - (b) Randomly initialize particle positions $\mathbf{x}_0^i \in \mathbf{D}$ for $i = 1, \dots, p$.
 - (c) Randomly initialize particle velocities $0 < \mathbf{v}_0^i < v_0^{max}$ for $i = 1, \dots, p$.
 - (d) Set $k = 1, i = 1$
2. Optimize
 - (a) Evaluate fitness value f_k^i using design space coordinates \mathbf{x}^i
 - (b) If $f_k^i < f_{best}^i$ then $f_{best}^i = f_k^i, \mathbf{p}_k^i = \mathbf{x}_k^i$
 - (c) If $f_k^g < f_{best}^g$ then $f_{best}^g = f_k^g, \mathbf{p}_k^g = \mathbf{x}_k^i$
 - (d) If stopping condition is satisfied go to 3.
 - (e) Update particle velocity vector \mathbf{v}_{k+1}^i using (2).
 - (f) Update particle position vector \mathbf{x}_{k+1}^i using (1).
 - (g) Increment i . If $i > p$ then increment k , set $i = 1$.
 - (h) Go to 2(a).
3. Report results and terminate

The above represents an improved variant of the original sequential algorithm [7], in which each particle's fitness evaluation is executed sequentially and the best position of the swarm is updated as soon as a particle finds a better position. The parallel implementation modifies step 2 of the above algorithm to evaluate all particles in parallel as follows:

2. Optimize
 - (a) Evaluate **all** fitness values f_k^i using **parallel processes**, using design space coordinates \mathbf{x}^i for $i = 1, \dots, p$.
 - (b) **Barrier synchronization (wait for all processes to finish).**
 - (c) If $f_k^i < f_{best}^i$ then $f_{best}^i = f_k^i, \mathbf{p}_k^i = \mathbf{x}_k^i$ for $i = 1, \dots, p$.
 - (d) If $f_k^g < f_{best}^g$ then $f_{best}^g = f_k^g, \mathbf{p}_k^g = \mathbf{x}_k^i$ for $i = 1, \dots, p$.

- (e) If stopping condition is satisfied go to 3.
- (f) Update particle velocity vector v_{k+1}^i using (2).
- (g) Update particle position vector x_{k+1}^i using (1).
- (h) Increment k.**
- (i) Go to 2(a).

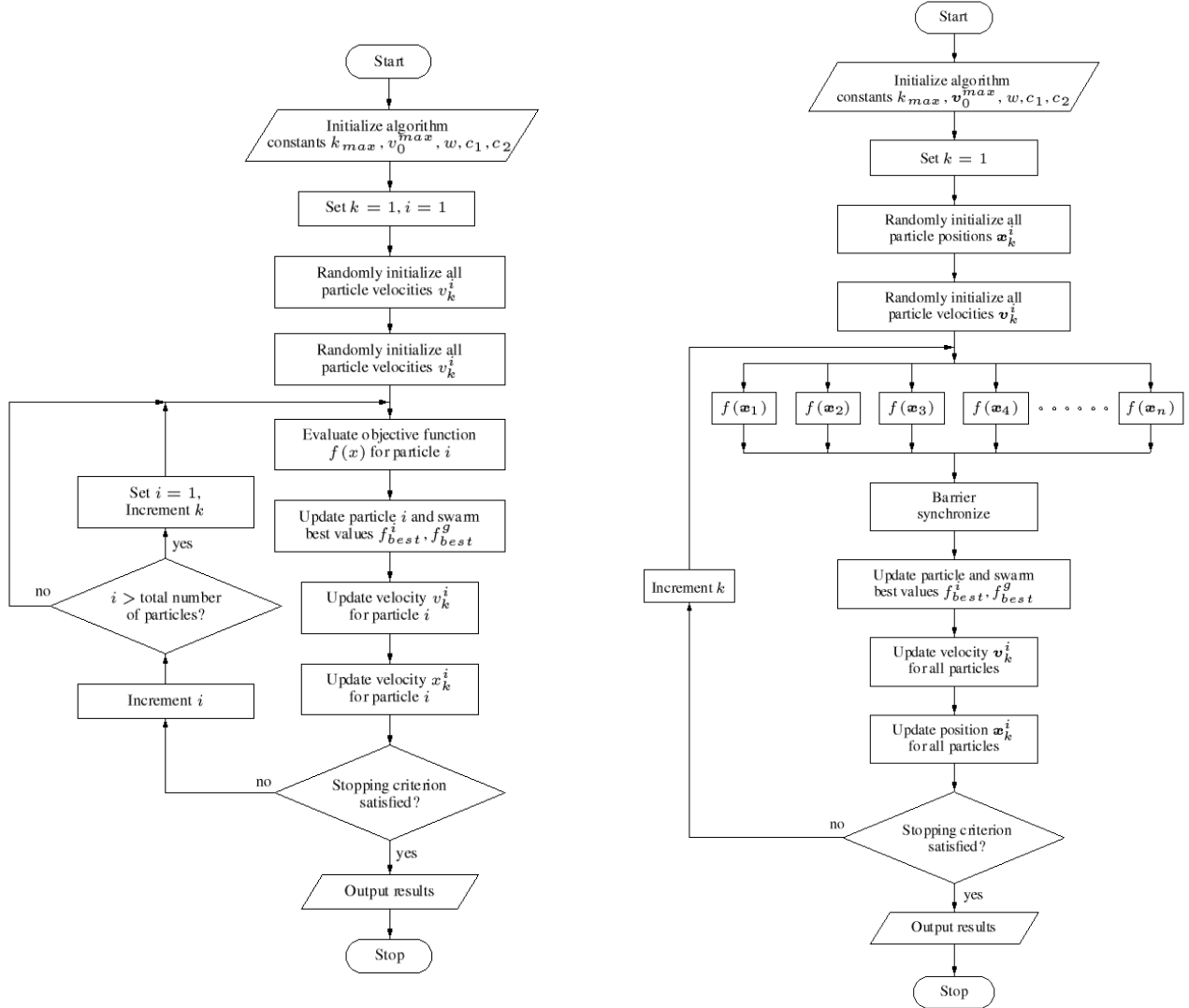


Figure 2. Original and parallelized particle swarm algorithm flow diagrams

The improvement of updating the cognitive f_k^i and social f_k^g best remembered positions, p_k^i and p_k^g , after each individual particle fitness evaluation is lost to us in the parallel implementation. This modification, which offered improved convergence rates, is impossible to duplicate in the parallel PSO implementation because of the barrier synchronization. This barrier synchronization stops the algorithm from proceeding to the next step until all of the objective functions have been reported to the master node, which is required to maintain algorithm coherence. The parallelization comes at the cost of delaying the update of the best swarm position until the entire swarm's fitness evaluations have been performed. The effect of this change on algorithm performance was investigated in [7,13], where it was found to be of the order of 30%. This may be viewed as one of the overhead costs associated with parallelization.

Application to system identification problem

Our large-scale optimization problem entails the system identification of joint angle locations and orientations from a set of marker trajectories. This procedure requires data measured by recording the trajectories of a set of markers, which are externally attached to the subject's skin, using a system of digital cameras with multiple viewing angles [8,9]. These recordings are processed to obtain trajectory data in a laboratory fixed coordinate system. Models obtained in this manner are used in forward or inverse dynamic simulations [5] to quantify the results of certain surgical procedures, or to aid in the design of prosthetic devices.

For the system identification procedure we first need to create a parametric kinematic linkage model (Figure 3) with a set of virtual markers, which correspond to actual markers adhered to the subject. This model needs to be able to emulate the limb segment movements by exhibiting an appropriate number of degrees of freedom, in our case two revolute joints. By optimizing the orientation and location parameters p_1 - p_{12} of this model, with the objective of minimizing the cumulative sum of square distance errors between the parametric model markers and actual marker trajectories (3,4), we are able to recover the original model configuration [11,12]. To evaluate the accuracy with which the PSO would be able to recover the original joint configuration, a kinematic linkage model with a set of known parameters was used to create a synthetic marker trajectory dataset. This dataset was then analyzed in the exact same manner as experimentally measured marker trajectories, and the parameter configuration compared to the original.

The marker distance error minimization with $n = 50$ trajectory time frames, and $m = 6$ markers is formulated as follows:

$$\min_p f(p) = \sum_{i=1}^n \sum_{j=1}^m \Delta_{i,j}^2 \quad (3)$$

with $\Delta_{i,j}$ being the distance between the experimental and analytical position of marker j in time frame i .

$$\Delta_{i,j}^2 = \Delta x_{i,j}^2 + \Delta y_{i,j}^2 + \Delta z_{i,j}^2 \quad (4)$$

In order to approximate experimental data more accurately noise was superimposed on the synthetic marker trajectories by using a sinusoidal noise model with a random frequency ($0 \leq \omega \leq 25 \text{ rad/sec}$), phase ($0 \leq \phi \leq 2\pi$) and amplitude ($0 \leq A \leq m$) based on a model by [10].

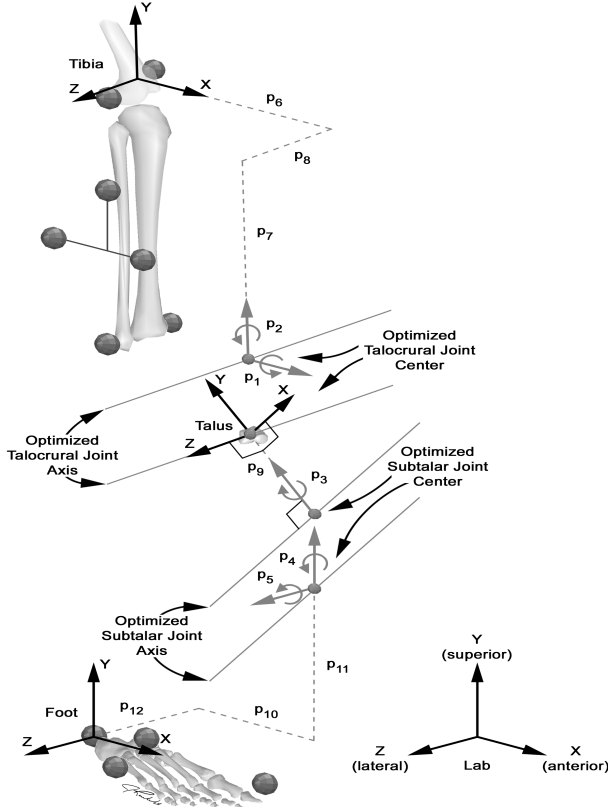


Figure 3. Parametric kinematic linkage model

6. Numerical results

For the numerical results all optimizations were performed using a swarm of 20 particles on a cluster of 1.3 GHz networked personal computers with the Linux operating system. As can be seen from Table 1 the algorithm had little difficulty recovering the original parameters from the synthetic data set (with no superimposed noise), with a final cumulative error value f in the order of 10^{-13} . This results in the optimum model being recovered with mean angular errors less than or equal to 0.045 degrees and mean position errors less than or equal to 0.0077 cm.

Parameter	Upper bound	Lower bound	Synthetic solution	Synthetic data	Synthetic data + noise
P1 (degrees)	48.66935	-11.633065	18.366935	18.364964	15.1301
P2 (degrees)	30	-30	0	-0.011809	8.0075
P3 (degrees)	70.230969	10.230969	40.230969	40.259663	32.9741
P4 (degrees)	53	-7	23	23.027088	23.12202
P5 (degrees)	72	12	42	42.00208	42.03973
P6 (cm)	6.270881	-6.270881	0	0.00027	-0.3936
P7 (cm)	-33.702321	-46.244082	-39.973202	-39.972852	-39.61422
P8 (cm)	6.27088	-6.270881	0	-0.000287	0.75513
P9 (cm)	0	-6.270881	-1	-1.000741	-2.81694
P10 (cm)	15.266215	2.724454	8.995334	8.995874	10.21054
P11 (cm)	10.418424	-2.123338	4.147543	4.147353	3.03367
P12 (cm)	6.888097	-5.653664	0.617217	0.616947	-0.19037

Table 1. Recovery of joint orientations and location from synthetic data and synthetic data with noise.

For the synthetic data with noise a mean marker error of 0.514485 was found, which is on the same order as the imposed numerical noise. This error corresponds to a mean angular error 3.732191 degrees and a mean position error of 0.923724 cm (Table 2).

Synthetic + noise	
Mean marker distance error (cm)	0.514485 ± 0.233956
Mean angular parameter error (degrees)	3.732191 ± 3.394553
Mean position parameter error (cm)	0.923724 ± 0.471443

Table 2. Synthetic data + noise recovered model marker errors with standard deviation.

The poor agreement of orientation parameters p_{1-4} to actual values for the noisy dataset is the result of the induced numerical noise. This can be explained by the dependency of angular calculations on marker positions. Because of the proximity of markers to each other, even relatively small mean amplitude numerical noise can result in large fluctuations in calculated limb angles. This is also observed in the mean angular parameter error in Table 2.

Performance results using our biomechanical system identification example problem are presented in Figure 4. A study was performed to investigate the effect of using an increasing amount of computational nodes to solve a fixed task size of 1000 fitness evaluations. The algorithm yields a significant increase in throughput as the number of nodes is increased. This increase is not linear however, due to increasing network communication overhead between nodes at higher number of processors. Using a varying population of particles has been shown to have little impact on cost [7], but the PSO's performance in terms of reliability may suffer at lower numbers of particles.

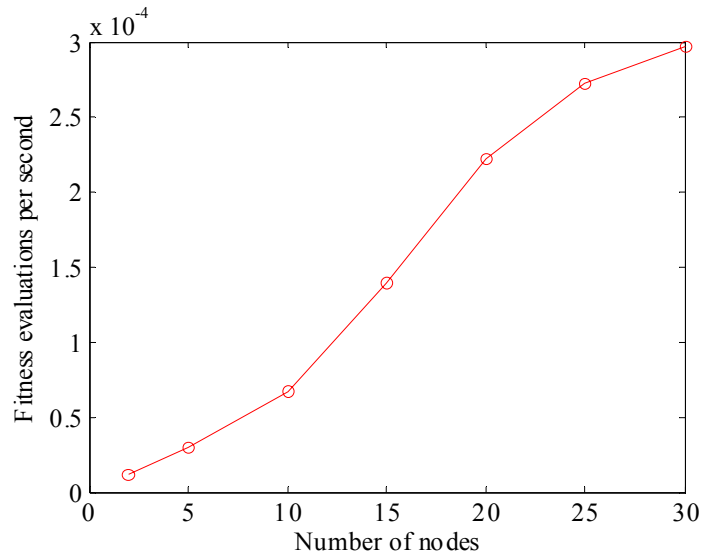


Figure 4. Optimization analysis speedup with an increasing amount of computational nodes (1000 fitness evaluations).

7. Conclusions

In this study, we presented a parallelization of the particle swarm optimization algorithm and applied it to system identification of kinematic skeletal models from marker trajectory data. The parallelization is based on master/slave processor model, and it requires a synchronization of the algorithm that may lead to small increase in required function evaluations. However, this small loss is recovered with a small number of processors. . By using a parallel computation approach the time required to solve the system identification problem was reduced substantially, proving that optimization using a parallel particle swarm algorithm on a cluster of processors is a viable and worthwhile option to solve large-scale optimization problems exhibiting multiple local minima. The method was demonstrated by accurately recovering the parameters on an ankle model from synthetically generated data with superimposed noise.

8. Acknowledgements

This study was funded by NIH National Library of Medicine (R03 LM07332-01) and Whitaker Foundation grants to B.J.F. and AFOSR grant F49620-09-1-0070 to R.T.H.

9. References

1. J. Kennedy and R.C. Eberhart, 1995, "Particle swarm optimization", Proc. IEEE International Conference on Neural Networks, Perth, Australia, Vol. 4, pp. 1942-1948.
2. P.C. Fourie and A.A. Groenwold, 2001, "Particle swarms in size and shape optimization", Proc. Workshop on Multidisciplinary Design Optimization, pp.97-106, Pretoria, South Africa.
3. A.J. van Soest and L.J.R. Casius., 2003 "The merits of a parallel genetic algorithm in solving hard optimization problems", J. Biomech. Eng., Vol. 125 , pp. 141-146.
4. B. Monien, F. Ramme, and H.Salmen, 1995, "A parallel simulated annealing algorithm for generating 3D layouts of undirected graphs", Proc. 3rd Int. Symp. Graph Drawing, GD, pp. 396-408, Berlin, Germany.
5. G. Venter, and B.C. Watson, "Exploiting parallelism in general purpose optimization", 2000, Proceedings, 6th International Conference on Applications of High-Performance Computers in Engineering, Maui, Hawaii.
6. M.G. Pandy, 2001, "Computer modeling and simulation of human movement", Annu. Rev. Biomed. Engr, Vol. 2, pp. 245-273.
7. A. Carlisle, G. Dozier, 2001, "An off-the-shelf PSO", Proc. Workshop on Particle Swarm Optimization, Purdue School of Engineering and Technology 2001, Indianapolis, USA.
8. A.J. van den Bogert, G.D. Smith and B.M. Nigg, 1994, "In vivo determination of the anatomical axes of the ankle joint complex: an optimization approach", J. Biomech. Vol 12, pp. 1477-88.
9. I. Soderkvist and P.A. Wedin, 1993, "Determining the movements of the skeleton using well-configured markers", J. Biomech., Vol 26, pp. 1473-77.
10. L. Cheze, B.J. Fregly and J. Dimnet, 1995. "A solidification procedure to facilitate kinematics analyses based on video system data", J. Biomech., Vol. 28, pp. 879-884.
11. C.W. Spoor and F.E. Veldpaus, 1980, "Rigid body motion calculated from spatial co-ordinates of markers", J. Biomech., Vol 13, pp. 391-393
12. T.-W Lu and J.J. O'Connor, 1999, "Bone position estimation from skin marker co-ordinates using global optimization with joint constraints", J. Biomech., Vol 32, pp. 129-134.
13. J.F. Schutte, 2001, "Particle Swarms in Sizing and Shape Optimization", Masters Thesis, University of Pretoria, South Africa.