LARGE AND PARALLEL PARTICLE SWARMS

by

Andrew McNabb

A thesis proposal submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

December 2008

1 Abstract

Particle Swarm Optimization (PSO) is a simple and effective evolutionary algorithm, but it may take hours or days to optimize difficult objective functions. Deceptive functions require extensive exploration to avoid trapping PSO in local optima. Computationally complex functions, which may depend on detailed simulations or large datasets, take a long time to evaluate. In both cases, PSO must be parallelized to use multiprocessor systems and clusters efficiently.

I propose to investigate the implications of parallelizing PSO and in particular, the details of parallelization and the effects of large swarms. PSO can be expressed naturally in Google's MapReduce framework to develop a simple and robust parallel implementation that automatically includes communication, load balancing, and fault tolerance. This flexible implementation would make it easy to apply modifications to the algorithm. Such modifications to PSO include changes to improve optimization of difficult objective functions and to improve parallel performance. Larger swarms help with both of these goals, but they are most effective if arranged into sparse topologies that reduce the overhead of communication. Additionally, since some objective functions need information to flow quickly through a swarm, PSO must be modified to use communication more efficiently in a large sparse swarm.

2 Introduction

Particle Swarm Optimization (PSO) is an optimization algorithm that was inspired by experiments with simulated bird flocking [6]. This evolutionary algorithm has become popular because it is simple, requires little tuning, and has been found to be effective for a wide range of problems.

Often a function that needs to be optimized takes a long time to evaluate. A problem using web content, commercial transaction information, or bioinformatics data, for example, may involve large amounts of data and require minutes or hours for each function evaluation. Each iteration of PSO is slow for such functions because the function must be evaluated sequentially for each particle in the swarm. Alternatively, some functions may be easy to evaluate but difficult to optimize. For example, high dimensional problems may take many iterations to converge, and functions with deceptive local optima may converge prematurely. In both of these cases, PSO must be parallelized to fully utilize resources in multiprocessor systems and supercomputing clusters.

Unfortunately, large-scale PSO, like all large-scale parallel programs, faces a wide range of problems. Inefficient communication or poor load balancing can keep a program from scaling to a large number of processors. Once a program successfully scales, it must still address the issue of failing nodes. For example, assuming that a node fails, on average, once a year, then the probability of at least one node failing during a 24-hour job on a 256-node cluster is $1-(1-1/365)^{256} = 50.5\%$. On a 1000-node cluster, the probability of failure rises to 93.6%.

Google faced these same problems in large-scale parallelization, with hundreds of specialized parallel programs that performed web indexing, log analysis, and other operations on large datasets. Engineers created a common system to simplify these programs. Google's MapReduce is a programming model and computation platform for parallel computing [**?**]. It allows simple programs to benefit from advanced mechanisms for communication, load balancing, and fault tolerance.

PSO can be expressed naturally in Google's MapReduce framework to avoid explicitly addressing any of the details of parallelization, such as communication, load balancing, and fault tolerance. This approach would make a simple, flexible, scalable, and robust implementation because it is designed in the MapReduce parallel programming model.

PSO has typically been used with small swarms of 50 particles [2]. But with the widespread availability of multicore processors and even a modest number of particles per core, swarms running on a cluster could easily have thousands of particles. Some have observed that with many processors, larger swarm sizes may improve the rate of convergence [14, 16].

2.1 Particle Swarm Optimization

Particle Swarm Optimization simulates the motion of particles in the domain of a function. These particles search for the optimum by evaluating the function as they move. During each iteration of the algorithm, the position and velocity of each particle are updated. Each particle is pulled toward the best position it has sampled (personal best) and the best position of any particle in its neighborhood (global best). This attraction is weak enough to allow exploration but strong enough to encourage exploitation of good locations and to guarantee convergence.

Each particle's position and velocity are initialized to random values based on a function-specific feasible region. During each iteration of constricted PSO, the following equations update a particle's position $\vec{x}$ and velocity $\vec{v}$ with respect to personal best $\vec{p}$ and global best $\vec{g}$:

$$\vec{v}_{t+1} = \chi \left[\vec{v}_t + \phi_1 \, \mathrm{U}() \otimes (\vec{p} - \vec{x}_t) + \phi_2 \, \mathrm{U}() \otimes (\vec{g} - \vec{x}_t)\right] \tag{1}$$

$$\vec{x}_{t+1} = \vec{x}_t + \vec{v}_{t+1} \tag{2}$$

where $\phi_1 = \phi_2 = 2.05$, U() is a vector of samples drawn from a standard uniform distribution, and $\otimes$ represents element-wise multiplication. The constriction constant χ is:

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|}$$

where $\phi = \phi_1 + \phi_2$ and κ is usually 1.0. [3]

The topology or sociometry of a swarm indicates how information is communicated between particles. The most typical representation of swarm topology is an undirected graph, where each vertex is a particle. The neighborhood of a particle—the set of vertices with shared edges—has two intuitive interpretations. From one perspective, it indicates which neighbors (or “informants”) contribute their personal bests to a particular particle’s global best. Alternatively, it indicates which neighbors a particular particle sends its personal best to. If the graph is undirected, these two perspectives are equivalent.

2.2 MapReduce

MapReduce is a functional programming model that is well suited to parallel computation. In this model, a program consists of two high-level functions, a map function and a reduce function, which meet a few simple requirements. If a problem is formulated in this way, it can be parallelized automatically. The MapReduce implementation manages the details of parallelization so that the user-specified program does not have to. [**?**]

In MapReduce, all data are in the form of key-value pairs. For example, in a program that counts the frequency of occurrences for various words, a key would be a word from the input text, and the corresponding value would represent the frequency of that word.

A MapReduce operation takes place in two main stages. In the first stage, the map function is called once for each input record. At each call, it may produce any number of output records. In the second stage, this intermediate output is sorted and grouped by key, and the reduce function is called once for each key. The reduce function is given all associated values for the key and outputs a new list of values (often “reduced” in length from the original list of values).

A map function is defined as a function that takes a single key-value pair and outputs a list of new key-value pairs. The input key may be of a different type than the output keys, and the input value may be of a different type than the output values:

$$\mathsf{map} : (K_1, V_1) \rightarrow \mathsf{list}((K_2, V_2))$$

Since the map function only takes a single record, all map operations are independent of each other and fully parallelizable.

A reduce function is a function that reads a key and a corresponding list of values and outputs a new list of values for that key. The input and output values are of the same type. Mathematically, this would be written:

$$\mathsf{reduce} : (K_2, \mathsf{list}(V_2)) \rightarrow \mathsf{list}(V_2)$$

A reduce operation may depend on the output from any number of map calls, so no reduce operation can begin until all map operations have completed. However, the reduce operations are independent of each other and may run in parallel.

The data given to the map and reduce functions are generally as fine-grained as possible. This ensures that the implementation can split up and distribute tasks. The MapReduce system consolidates the intermediate output from all of the map tasks. These records are sorted and grouped by key before being sent to the reduce tasks.

Although not all algorithms can be efficiently formulated in terms of map and reduce functions, MapReduce provides many benefits over other parallel processing models. In this model, a program consists of only a map function and a reduce function. The infrastructure provided by a MapReduce implementation manages all of the details of communication, load balancing, fault tolerance, resource allocation, job startup, and file distribution. This runtime system is written and maintained by parallel programming specialists, who can ensure that the system is robust and optimized, while those who write mappers and reducers can focus on the problem at hand without worrying about implementation details.

A MapReduce system determines task granularity at runtime and distributes tasks to compute nodes as processors become available. If some nodes are faster than others, they will be given more tasks, and if a node fails, the system automatically reassigns the interrupted task.

3 Related Work

There are several parallel adaptations of Particle Swarm Optimization. As with my proposal, Synchronous PSO [16] and Global PSO [1] preserve the exact semantics of serial PSO. In contrast, Parallel Asynchronous PSO does not preserve the exact semantics of serial PSO, but instead focuses on better load balancing [7]. Other variants propose different topologies to limit communication among particles and between groups of particles [1, 12]. Parallel PSO has been applied to applications including antenna design [4] and biomechanics [7] and has been adapted to solve multiobjective optimization problems [12, 13].

Google has described its MapReduce implementation in published papers [**?**] and slides, but it has not released the system to the public. Presumably the implementation is highly optimized because Google uses it to produce its web index. The Apache Lucene project has developed Hadoop, an Java-based open-source clone of Google's closed MapReduce platform. The platform is relatively new but rapidly maturing. At this time, Hadoop overhead is significant but not overwhelming and is expected to decrease with further development.

Particle swarms are usually small, with about 50 particles in either the complete (fully-connected) or ring topology [2, 9]. Although simple static topologies are most common, dynamic and adaptive topologies include TRIBES [**?**], stochastic star [10], an increasingly connected ring [15], Randomized Directed Neighborhoods [11], and Dynamic Multi-Swarm [8].

With the widespread availability of multicore processors and even a modest number of particles per core, swarms running on a cluster could easily have thousands of particles. Large particle swarms have received little attention, but it has been observed that larger swarm sizes may improve the rate of convergence [14, 16], although adding particles to the "fully-informed PSO" has been less successful [**?**]. Dividing a large swarm into smaller subswarms is an alternative to using a single global swarm [5, 8].

4 Thesis Statement

Particle Swarm Optimization can be expressed in the MapReduce model to provide a flexible and robust parallel implementation. Large swarms, which work particularly well in this implementation, improve PSO performance for some types of objective functions. To achieve the best parallel performance for many of these functions, communication must be reduced by using specialized topologies and by modifying the algorithm to accelerate the flow of information through the swarm.

5 Project Description

The project will consist of three major stages. The first stage will consider the details of implementing PSO as a MapReduce program and will consider the parallel performance of this implementation. The second stage will investigate the implications of large swarms of thousands of particles and determine the effectiveness of PSO on several benchmark functions. The third stage will evaluate PSO with very large swarms of hundreds of thousands of particles and explore the behavior of subswarms.

5.1 MapReduce Particle Swarm Optimization

The first stage of the project will reformulate Particle Swarm Optimization as a sequence of map and reduce functions. It will evaluate the performance of this parallel algorithm and determine when this is an effective approach.

In an iteration of Particle Swarm Optimization, each particle in the swarm moves to a new position, updates its velocity, evaluates the function at the new point, updates its personal best if this value is the best seen so far, and updates its global best after comparison with its neighbors. Except for updating its global best, each particle updates independently of the rest of the swarm.

Due to the limited communication among particles, updating a swarm can be formulated as a MapReduce operation. As a particle is mapped, it receives a new position, velocity, value, and personal best. In the reduce phase, it incorporates information from other particles in the swarm to update its global best. This implementation conforms to the MapReduce model while performing the same calculations as standard Particle Swarm Optimization.

5.2 Large Swarms

Parallelism is usually most effective when a problem can be split into fine-grained tasks. In the case of parallel PSO in MapReduce, decomposability is improved by increasing the number of particles in a swarm. This raises the question of how PSO performs with large particle swarms of thousands of particles.

The ideal topology and swarm size for Particle Swarm Optimization depend on the objective function. Mendes [9] compared the performance of PSO with a number of different benchmark functions under the assumption that the benchmark functions represent broad classes of interesting functions. Bratton and Kennedy [2] found no significant difference in performance for any swarm sizes between 20 and 100, but Perez and Basterrechea [14] and Schutte et al. [16] found that large swarms could improve performance for some functions.

The second stage of the project will compare the performance of PSO on three benchmark functions with a variety of topologies, especially those with conservative levels of communication. This will provide some basic observations about how the choice of topology affects performance with a large swarm. It will also consider algorithmic changes to accelerate the flow of information through a swarm. Conclusions about which approaches are most effective will point out appropriate directions for the third stage of the project.

The Sphere, Rastrigin, and Griewank benchmark functions are representative of three different types of functions. Sphere is unimodal and smooth. It is most easily optimized when information flows between particles as quickly as possible. In the case of Rastrigin, which is much less smooth, PSO tends to prematurely converge to local minima. However, increasing communication still tends to improve performance for this function, for which PSO performs best with fully connected swarms. Griewank has even more deceptive local minima and is poorly optimized if information flows too quickly through the swarm.

5.3 Massively Parallel Swarms

The third stage will build upon the results of Section 5.2 and evaluate the performance of PSO with even larger swarms of hundreds of thousands of particles. One interesting approach is to split a very large swarm into smaller subswarms. Each subswarm will operate independently for some number of iterations before communicating its results with other subswarms. This approach will allow for an extremely large number of particles while reducing the communication and its associated overhead.

Additionally, this stage will evaluate swarms on various benchmark functions and a wider range of swarm sizes. This will provide a more definitive resource for determining which topologies are most effective for various types of objective functions.

6 Validation

The project is expected to produce a practical parallel implementation of PSO, to show the effectiveness of large swarms for deceptive functions, and to propose topologies and/or variants of PSO that improve performance without increasing communication costs.

6.1 Parallel PSO

The first stage of the project is successful if it produces a practical parallel implementation of PSO. This can be demonstrated by experiments that show acceptable performance for difficult functions. For functions that take a long time to evaluate, a parallel implementation must be scalable to dozens of processors, even without large swarms. Parallel performance should be measured with speedup, a common empirical measure of scalability.

Speedup is defined as the ratio of the serial runtime t_1 of the best sequential algorithm to the time t_p taken by the parallel algorithm to solve the same problem on p processors [?]:

$$S_p = \frac{t_1}{t_p} \tag{3}$$

The definition of speedup is ambiguous as to what constitutes the best sequential algorithm, and this ambiguity can make speedup graphs difficult to interpret. To avoid detailed arguments about whether it is the best implementation of PSO, the sequential implementation should implement a variant that is considered standard. Since the MapReduce implementation does not change the PSO algorithm itself, it can build on the sequential implementation and use the same language and architecture.

6.2 Large Swarms

The second stage of the project will study the behavior of PSO with thousands of particles and with various topologies. This stage is successful if it identifies situations where large particle swarms perform better than small swarms. However, large swarms are not expected to outperform small swarms for all benchmark functions.

Variants and topologies of PSO are usually evaluated by comparing the best value vs. the number of function evaluations needed to achieve that value. In a parallel implementation, it may be possible to perform additional function evaluations in parallel without increasing the overall execution time. In this case, it may be more appropriate to evaluate PSO by considering the best value vs. the number of iterations. When comparing a wide range of swarm sizes, it is also be useful to see the best value after a fixed number of iterations vs. the number of particles in the swarm.

6.3 Very Large Swarms

The third stage investigates very large swarms. While the issues from the second stage still apply, communication becomes extremely important when there are hundreds or thousands of particles per processor. In this case, the execution time of parallel PSO can be improved by reducing the amount of communication. This stage is successful if finds practical topologies that reduce communication without significantly diminishing performance.

Communication costs may vary from one platform to another, so topologies should be evaluated by the number of messages required rather than the wall clock time. The time complexity of any MapReduce implementation is $O(m \log m)$ in the number of messages. For PSO, the number of messages for n particles is at least n, since the state for each particle must be passed between tasks. Depending on the topology, the total number of messages is between $O(n)$ and $O(n^2)$, so the time required for communication overhead varies between $O(n \log n)$ and $O(n^2 \log n^2)$. A topology can be evaluated by the amount of communication it requires, but for many functions, reducing communication may impair the performance of PSO. To determine the best topology to use in a parallel implementation of PSO, we must compare the performance of various topologies using the same amount of communication. This will show which topologies use communication most effectively and to what degree PSO performance must be sacrificed to reduce communication.

7 Thesis Schedule

- Completed: reformulate PSO in MapReduce.
- February 6, 2009: evaluate PSO with large swarms with a few benchmark functions.
- February 20, 2009: evaluate PSO with very large swarms with subswarms and with more benchmark functions.
- March 9, 2009: rework individual papers into a coherent thesis and submit draft to committee chair.

- March 20, 2009: submit thesis to second committee member for review.
- March 30, 2009: submit thesis to third committee member for review.
- April 10, 2009: defend thesis.

Each of the three stages of the project can be submitted as a paper to a conference such as GECCO or CEC.

8 Annotated Bibliography

[1] M. Belal and T. El-Ghazawi. Parallel Models for Particle Swarm Optimizers. *Intl. Journal of Intelligent Computing and Information Sciences*, 4(1):100–111, 2004.

KEY: belal-2004-parallel-models-for-pso
ANNOTATION: Andrew: Applies techniques from parallel Genetic Algorithms (GAs) to Particle Swarm Optimization. It reviews global GAs, migration GAs, and diffusion GAs and introduces their PSO counterparts. For each parallel variant of PSO, it discusses the operation and message complexity. Although the paper does very little to discuss performance and draws few conclusions, it does a good job of showing the overall strategies for parallelizing PSO.

[2] D. Bratton and J. Kennedy. Defining a Standard for Particle Swarm Optimization. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pp. 120–127. 2007.

KEY: bratton-2007-defining-a-standard-for-pso
ANNOTATION: Andrew: Proposes a new canonical definition of PSO based on the state of the art. It recognizes that Constriction PSO has been shown to consistently outperform the original algorithm and recommends that it be treated as the standard form. It also considers topologies, swarm sizes, initialization, and boundary conditions and makes recommendations in each of these areas. The intent is not to limit research but rather to establish a common standard to compare future variants against. This paper does a good job of addressing an important problem.

[3] Maurice Clerc and James Kennedy. The Particle Swarm—Explosion, Stability, and Convergence in a Multidimensional Complex Space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

Key: clerc-2002-constricted-pso

Annotation: Andrew: Analyzes particle motion in PSO from algebraic (discrete) and analytic (continuous) points of view. The paper proposes restricting velocity with a constriction coefficient to guarantee convergence. This variant, known as Constriction PSO, is now the most common form of the PSO motion equations.

Matt: General particle swarm article - talks about the constriction factor. I believe it's the original proof of convergence and the need of the constriction factor.

[4] Nanbo Jin and Yahya Rahmat-Samii. Parallel Particle Swarm Optimization and Finite-Difference Time-Domain (PSO/FDTD) Algorithm for Multiband and Wide-Band Patch Antenna Designs. *IEEE Transactions on Antennas and Propogation*, 53(11):3459–3468, 2005.

Key: jin-2005-pso-antenna-designs

Annotation: Andrew: Applies a parallel implementation of PSO, using a master-slave model, to an antenna design problem.

[5] Johannes Jordan, Sabine Helwig, and Rolf Wanka. Social Interaction in Particle Swarm Optimization, the Ranked FIPS, and Adaptive Multi-Swarms. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pp. 49–56. ACM, 2008.

Key: jordan-2008-social-interaction-in-pso-fips-adaptive

Annotation: Andrew: Compares the performance of several topologies and communication strategies to address several drawbacks of Fully Informed PSO. The paper considers variants based on the Von Neumann topology, stereotyping, FIPS, and completing graphs. It also considers using subswarms with each of these.

[6] James Kennedy and Russell C. Eberhart. Particle Swarm Optimization. In *International Conference on Neural Networks IV*, pp. 1942–1948. Piscataway, NJ, 1995.

Key: kennedy-1995-particle-swarm-optimization

Annotation: Andrew: Introduces Particle Swarm Optimization and the process through which it was developed.

[7] Byung-Il Koh, Alan D. George, Raphael T. Haftka, and Benjamin J. Fregly. Parallel Asynchronous Particle Swarm Optimization. *International Journal of Numerical Methods in Engineering*, 67:578–595, 2006.

Key: koh-2006-parallel-asynchronous-pso

Annotation: Andrew: Introduces an asynchronous parallel implementation of PSO, where particles iterate independently rather than in lock-step. Particles do not wait for communication from neighbors. This is particularly effective if compute clusters are large or heterogeneous or if the speed of function evaluation varies.

[8] J.J. Liang and P.N. Suganthan. Dynamic Multi-Swarm Particle Swarm Optimizer. In *Proceedings of the 2005 Swarm Intelligence Symposium*, pp. 124–129. 2005.

Key: liang-2005-dynamic-multi-swarm-pso

Annotation:

[9] Rui Mendes. *Population Topologies and Their Influence in Particle Swarm Performance.* Ph.D. thesis, Escola de Engenharia, Universidade do Minho, 2004.

Key: mendes-2004-population-topologies-in-pso

Annotation: Andrew: Investigates the importance of topologies in PSO performance. The dissertation describes a variety of optimization algorithms, along with a thorough survey of PSO and its variants. It then reviews several graph statistics that can be used to characterize particle swarm topologies. Finally, it explores the importance of these graph statistics by showing the performance of various auto-generated topologies with six benchmark functions.

[10] Vladimiro Miranda, Hrvoje Keko, and Álvaro Jaramillo Duque. Stochastic Star Communication Topology in Evolutionary Particle Swarms. *International Journal of Computational Intelligence Research*, 4(2):105–116, 2008.

Key: miranda-2008-stochastic-star-evolutionary-pso

Annotation: Andrew: Presents a dynamic random topology to reduce communication within the context of the EPSO variant. In this topology, each particle randomly determines whether or not to use the global best during each iteration.

Matt: This paper talks mostly about EPSO, which adapts weights for which information source to use, but it mentions a stochastic sociometry. They don't make the same point we do, as they try to put it in the light of having an adaptive sociometry, but we should at least mention that random sociometry has been tried before. They don't, however, make the point that you need to remember your gbest or performance is horrible - they say "In other words, the information broadcasted in each iteration and in a particular dimension about the location of the global best could be lost of become corrupted with partial loss within the communication channel between particles." They don't remember the gbest.

[11] Arvind S. Mohais, Christopher Ward, and Christian Posthoff. Randomized Directed Neighborhoods with Edge Migration in Particle Swarm Optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pp. 548–555. 2004.

KEY: mohais-2004-randomized-directed-neighborhoods-in-pso
ANNOTATION: Andrew: Presents a dynamic topology with randomly generated neighborhoods and directed edges. At the end of each iteration, one randomly selected particle trades an edge with another randomly selected member of the swarm.

[12] Sanaz Mostaghim, Jürgen Branke, and Hartmut Schmeck. Multi-Objective Particle Swarm Optimization on Computer Grids. Technical Report 502, AIFB Institute, 2006.

KEY: mostaghim-2006-multi-objective-pso-on-grids
ANNOTATION: Andrew: Addresses parallel multi-objective PSO using subswarms with migration between processors.

[13] K. E. Parsopoulos, D. K. Tasoulis, and M. N. Vrahatis. Multiobjective Optimization Using Parallel Vector Evaluated Particle Swarm Optimization. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications*, pp. 823–828. 2004.

KEY: parsopoulos-2004-parallel-vector-evaluated-pso
ANNOTATION: Andrew: Describes a parallel implementation of Vector Evaluated PSO for multiobjective optimization.

[14] J.R. Perez and J. Basterrechea. Particle Swarm Optimization for Antenna Far-Field Radiation Pattern Reconstruction. In *Proceedings of the 36th European Microwave Conference*, pp. 687–690. 2006.

KEY: perez-2006-pso-radiation-application
ANNOTATION: Andrew: Applies PSO to an antenna problem and uses moderately large swarms of about 1000 particles.

[15] Mark Richards and Dan Ventura. Dynamic Sociometry in Particle Swarm Optimization. In *Proceedings of the Sixth International Conference on Computational Intelligence and Natural Computing*, pp. 1557–1560. 2003.

Key: richards-2003-dynamic-sociometry-in-pso

Annotation: Andrew: Proposes using completing graphs to balance exploration and exploitation. Completing graphs begin with a sparse topology and gradually add edges until the swarm is fully connected.

[16] J. F. Schutte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka, and A. D. George. Parallel Global Optimization with the Particle Swarm Algorithm. *International Journal for Numerical Methods in Engineering*, 61(13):2296–2315, 2004.

Key: schutte-2004-parallel-global-optimization-with-pso

Annotation: Andrew: Introduces a parallel implementation of PSO using MPI with one particle per processor. The benchmark functions used include a half-second delay for each evaluation, based on the observation that parallelization is inappropriate if function evaluations are fast.

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis proposal proposal submitted by

Andrew McNabb

This thesis proposal proposal has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Kevin Seppi, Chair
Date	Dan Ventura
Date	Mark Clement
Date	Kent Seamons Graduate Coordinator