An Asynchronous Hybrid Genetic-Simplex Search for Modeling the Milky Way Galaxy using Volunteer Computing

Genetic Algorithms Track

Travis Desell, Boleslaw K. Szymanski and Carlos Varela, RPI, Troy, NY
Genetic and Evolutionary Computing Conference, GECCO 2008, Atlanta, Georgia, July 12-16, 2008.

ABSTRACT

This paper examines the use of a probabilistic simplex operator for asynchronous genetic search on the BOINC volunteer computing framework. This algorithm is used to optimize a computationally intensive function with a continuous parameter space: finding the optimal fit of an astronomical model of the Milky Way galaxy to observed stars. The asynchronous search using a BOINC community of over 1,000 users is shown to be comparable to a synchronous continuously updated genetic search on a 1,024 processor partition of an IBM BlueGene/L supercomputer. The probabilistic simplex operator is also shown to be highly effective and the results demonstrate that increasing the parents used to generate offspring improves the convergence rate of the search. Additionally, it is shown that there is potential for improvement by refining the range of the probabilistic operator, adding more parents, and generating offspring differently for volunteered computers based on their typical speed in reporting results. The results provide a compelling argument for the use of asynchronous genetic search and volunteer computing environments, such as BOINC, for computationally intensive optimization problems and, therefore, this work opens up interesting areas of future research into asynchronous optimization methods.

Keywords

Genetic Algorithms, Simplex Method, Asynchronous Computing, Volunteer Computing

1. INTRODUCTION

In many scientific disciplines, the rate of data acquisition and model complexity is far outpacing developments in computing speed. As a result, distributed computing is becoming essential in advancing scientific discoveries by making complex modeling of large data sets possible in a reasonable amount of time. Volunteer computing platforms such as BOINC, can be extremely powerful computing tools, as has been shown by projects such as SETI@Home [5] and

Folding@Home [18], because they have the potential to provide millions of processors [6] at a fraction of the cost of supercomputing environments. However, in contrast to supercomputers, volunteer computing comes with significant challenges to overcome, namely scalability in the presence of higher heterogeneity and volatility. Volunteered computers can be of any speed and can disconnect randomly, possibly never to reappear. Because of this, traditional synchronous programming methods and algorithms are innapropriate.

A wide range of parallel genetic algorithms (PGAs) have been examined for different distributed computing environments. Generally, there are three types of parallel genetic algorithms: single population (panmictic, coarse-grained), multi-population (island, medium-grained), or cellular (fine-grained) [9]. Panmictic GAs create a population, evaluate it in parallel, and use the results to generate the next population. Island approaches evaluate local populations for a certain number of iterations, then exchange the best members with other islands [4, 7, 14, 17, 13]. Cellular algorithms evaluate individual parameter sets, then update these individual sets based on the fitness of their neighbors [3, 12]. Hybrid-granularity approaches have also been examined [16, 23].

Unfortunately, these approaches do not scale to the thousands of processors offered by the BOINC computing project. For these approaches, scalability is limited to the size of the population being used and the parallelizability of the fitness calculation. Additionally, they are not fault tolerant - if any processor becomes unavailable, the search cannot proceed and will fail. Finally, while load balancing and partitioning is feasible on computational grids [16, 14], it becomes impractical or impossible when there are no clusters of homogeneous processors and failures are frequent. Desell et al. have introduced an asynchronous genetic search framework that is both scalable and fault tolerant [11]. It uses a single population, and asynchronously generates new members to be evaluated when work is requested and continuously updates the population by inserting members when their fitness is reported. They have also shown that using different operators as part of this asynchronous genetic search can significantly improve the convergence rate to the global optimum.

This paper investigates the use of a new crossover operator based on the Nelder-Mead simplex algorithm and compares it to the double shot operator introduced by Desell et al for an astronomical modeling application. This application fits models of the Milky Way galaxy to stars observed by the SLOAN digital sky survey [19, 1]. Due to the expense of

Algorithm 1: Asynchronous Report Work

Data: P /*Population*/,
max /*Maximum Population Size*/,
R /*Result*/
Result: Updated Population
if *P.size < max* **then**
  P.insert(R)
else if *R.fitness < worst(P).fitness* **then**
  P.insert(R)
  P.remove(worst(P))

Algorithm 2: Asynchronous Request Work

Data: P /*Population*/,
C /*Reproduction Probability*/,
max /*Maximum Population Size*/
Result: New Parameters to Evaluate
if *P.size < max* **then**
  return random_params();
else
  if *random() < C* **then**
    p1 = P[random()]
    p2 = P[random()], where p1 != p2
    return reproduce(p1, p2)
  else
    return mutate(P[random()])

---

calculating a single model (approximately 15 minutes to an hour for a small wedge of observed stars), finding the optimal fit in any reasonable amount of time requires a large amount of processing power. Because of this, the asynchronous search uses a BOINC-based computing project. Additionally, to measure the impact of asynchronicity and heterogeneity, the search is also tested synchronously on an IBM BlueGene/L supercomputer.

Results show that using the simplex crossover operator provides significant improvement in the convergence rates of the asynchronous genetic search compared to the double shot method. Additionally, utilizing asynchronous genetic search on BOINC is shown to be competitive to continuously updated synchronous genetic search on the BlueGene supercomputer. Analysis of the effects of heterogeneity and asynchronous reporting of member fitness shows that there is distinct possibility to further improve convergence rates through adaptive generation of new members and refinement of the space into which new members are generated. We feel that these results provide a strong argument for the use of asynchronous search on volunteer computing environments to perform computationally expensive scientific model optimization and that such searches are a viable alternative to their synchronous counterparts (which can only perform well on vastly more expensive computing environments).

This paper proceeds as follows. Section 2 discusses the asynchronous genetic search used. The double shot and simplex operators are presented in relation to other hybrid genetic search strategies in Section 3. Results detailing the convergence rates of the genetic search on both BOINC and the BlueGene are given in Section 4 along with an analysis of the effect of heterogeneity and asynchrony on the quality of members generated by these operators. The paper closes with conclusions and directions for future work in Section 5.

2. ASYNCHRONOUS GENETIC SEARCH

Asynchronous genetic search (AGS) is similar to traditional iterative genetic search (IGS) in that it keeps a population of parameters and generates reproductions and mutations based on it. However, instead of using a parallel model of concurrency like IGS, it uses a master-worker approach. Rather than iteratively generating new populations, new members of the population are generated when a worker requests work and the population is updated whenever work is reported to the master. The AGS algorithm consists of two phases and uses two asynchronous message handlers (see Algorithms 1 and 2). The actor model of computation [2] is assumed, so the server can either be processing a *request work* or a *report work* message and cannot process multiple messages at the same time.

In the first phase of the algorithm (while the population size is less than the maximum population size) the server is being initialized and a random population is generated. When a *request work* message is processed, a random parameter set is generated, and when a *report work* message is processed, the population is updated with the parameters and the fitness of that evaluation. When enough *report work* messages have been processed, the algorithm proceeds into the second phase which performs the actual genetic search.

In the second phase, *report work* will insert the new parameters and their fitness into the population but only if they are better than the worst current member and remove the worst member if required to keep the population size the same. Otherwise the parameters and the result are discarded. Processing a *request work* message will either return a mutation or reproduction (crossover) from the population.

This algorithm has significant benefits in heterogeneous environments because the calculation of fitness can be done by each worker concurrently and independently of each other. The algorithm progresses as fast as work is received, and faster workers can processes multiple *request work* messages, in the style of Cilk's work stealing [8], without waiting on slow workers. This algorithm is also highly scalable, as the only limiting factor is how fast results can be inserted into the population and how fast *request work* messages can be processed. It is also possible to have multiple masters using an island approach for even greater scalability.

3. HYBRID METHODS AND CROSSOVER OPERATORS

Most hybrid approaches to genetic search involve performing the genetic search in tandem with the hybrid. Chelouah and Siarry use iterations of genetic search to perform *diversification* until stopping conditions are met, then *intensification* on the best point found, repeating these two steps and achieved promising results on a set of classical test functions [10]. Satapathy et al. use a similar approach for image clustering, where the best $N+1$ points in the population are used to generate the simplex, instead of generating a simplex around the best point [21, 15]. Yen et al. use the same strategy in modeling metabolic systems, and examine some variations on reflecting the worst point through the centroid (reflecting the worst through the best point and the centroid
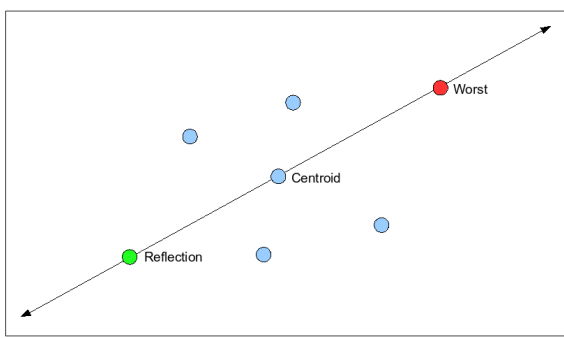
Figure 1: The simplex method takes the worst point and reflects it through the centroid of the remaining points. The probabilistic simplex operator randomly generates a point on some section of the line connecting the worst point and its reflection.
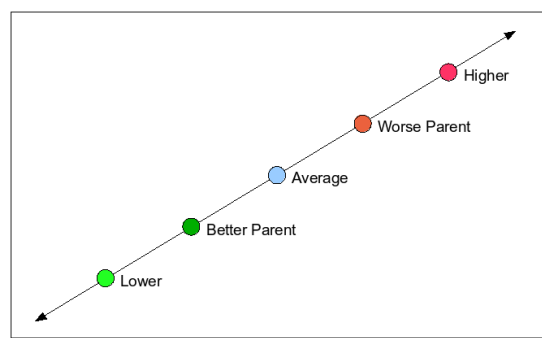

Figure 2: The double shot operator takes two parents and generates three children: the average of the parents, a point outside the worse parent (higher), and a point outside the better parent (lower), the latter two points are a distance from the average equal to the distance between their parents.

through the best point) [25]. Wei et al. apply this strategy within niches created after a set number of genetic search iterations by partitioning of the population, and performing the simplex search within each [24].

Renders and Bersini propose two methods to hybridize genetic search and hill climbing methods [20]: interleaving genetic search with hill-climbing and using hill climbing to create new crossover operators for the genetic search. The interleaving approach performs iterations of the simplex search on each individual of the population per iteration. However, they focus on utilizing a simplex crossover operator in addition to an average crossover operator and mutation. The simplex crossover operator selects $N + 1$ members of the population and performs an iteration of the simplex search - first attempting reflection, then expanding or contracting iteratively. Both hybrids are shown to outperform the non-hybrids, with the crossover hybrid performing the best. In future work, Seront and Bersini propose a mixture of these two methods, utilizing both interleaving and a simplex crossover operator [22], for even better performance.

3.1 Probabilistic Simplex Operator

Unfortunately, all of these approaches require synchrony by creating dependence between fitness calculations. While is it not possible to effectively perform the traditional Nelder-Mead simplex search in a highly heterogeneous and volatile environment like BOINC, a probabilistic operator can mimic its behavior. The Nelder-Mead simplex search takes $N + 1$ sets of parameters, and performs *reflection*, *contraction* and *expansion* operators between the worst set of parameters and the centroid of the remaining N (see Figure 1). After calculating the centroid, a line search is performed by expanding or contracting the simplex along this line. Because in our asynchronous model it is not possible to iteratively perform expansions and contractions, a random point is selected on the line joining the worst point and its reflection. There are three parameters involved in this operator, N, the number of points used to form the simplex (chosen randomly from the population), and two limits l_1 and l_2 which specify where on the line the point can be generated. For example, $l_1 = -1$ would set one limit to the reflection and $l_2 = 1$ would set the other limit to the worst point. For the purposes of this study, we use $l_1 = -1.5$ and $l_2 = 1.5$ and

examine how children generated from different parts of this line effect the evolution of the population.

3.2 Double Shot Operator

Desell et al. [11] show that using a double shot operator as opposed to a standard average operator can significantly improve convergence rates for the astronomical modeling application. The double shot operator produces three children instead of one. The first is the average of the two parents, and the other two are located outside the parents, equidistant from the average (see Figure 2). This approach is loosely based on line search, the point outside the more fit parent is in a sense moving down the gradient, while the point outside the less fit parent is moving up the gradient created by the two parents. The motivation for the latter point is to escape local minima.

4. RESULTS

4.1 Convergence

The hybrid simplex method was evaluated using the astronomical modeling problem detailed by Purnell et al [19]. Performing the evaluation of a single model to a small wedge of the sky consisting of approximately 200,000 stars can take between 15 minutes to an hour on a single high end processor. Because of this, to be able to determine the globally optimal model for that wedge in any tractable amount of time requires extremely high powered computing environments. To measure the effect of asynchronicity on the hybrid genetic search, both synchronous and asynchronous computing environments are used, 1024 processors of an IBM BlueGene/L and a BOINC volunteer computing project with over 1,000 volunteered computers.

Figure 3 shows the performance of the double shot approach, and the simplex approach with varying numbers of parents N being used to calculate the centroid on both environments. For both computing environments, a population of size 300 was used, and the mutation operator was applied 20% of the time, all other members were generated with the corresponding operator. The range of the probabilistic line search for the simplex hybrid was defined by the
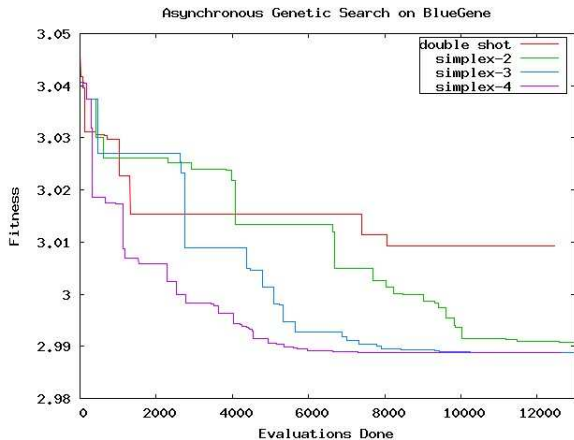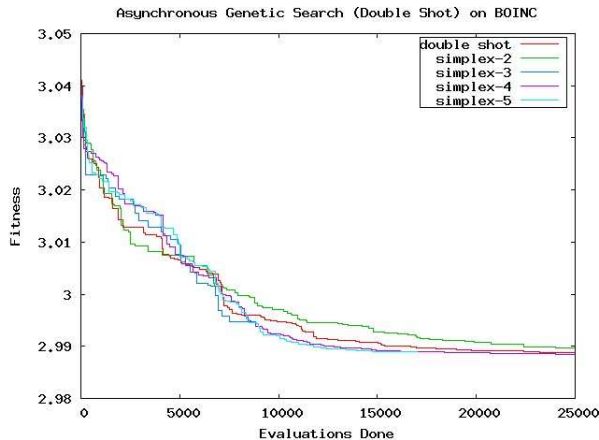
Figure 3: Fitness of the best member found averaged over five searches for the double shot approach and the simplex hybrid with $N = 2..5$, using the BOINC volunteered computers and the BlueGene supercomputer.

limits $l_1 = -1.5$ and $l_2 = 1.5$. For the synchronous execution on the BlueGene, each model evaluation was performed by dividing the work over the 1024 processors, and immediately attempting to insert the member into the population - in this way only the most evolved population was used to generate new members and the population was continuously updated. Previous work by Desell et al. [11] has shown that continuous update with the double shot approach significantly improves the convergence rate to the global minimum as compared to iterative traditional genetic search and continuously updated genetic search using only the average mutation operators. The asynchronous execution on BOINC generates new members from the current population whenever users request more work. After a user has completed the evaluation of a member, it's sent to the server and inserted into the population. There is no guarantee of when the fitness of a generated member will be returned, or even if it will be returned at all.

On the BlueGene, the hybrid simplex method shows dramatic improvement over the double shot approach, with the difference increasing as more parents are used to calculate the centroid. While the double shot method typically converges in around 18,000 iterations, the simplex hybrid with $N = 4$ converges in approximately 8,000. Compared to the 50,000 iterations reported for traditional iterative genetic search [11], the convergence rate is excellent. Using BOINC shows similar results, however the convergence rates are not as fast on the BlueGene, which is to be expected. Generally, increasing the number of points used to calculate the centroid results in better searches, however on BOINC the simplex with $N = 2$ and double shot operators initially seem to converge more quickly than the more informed simplex with $N = 3..5$, which was not the case on the BlueGene. The asynchronous approach on BOINC may take more iterations, but BOINC is much more accessible as it is dedicated to the project at hand, while use of the BlueGene is shared among many researchers. Because of this, even though the quantity of fitness evaluations done per second is similar for both computing environments, the BOINC framework can perform more searches and does so at a fraction of the cost. These results are very promising for the use of asynchronous

search and volunteer computing for computationally intensive scientific modeling.

4.2 Operator Analysis

To better understand the effect of the operators in evolving the population, as well as the effect of asynchronicity and of a highly heterogeneous computing environment on the fitness returned, the number of members processed between the generation and reporting of a members fitness was tracked, as well as information about how it was generated. For both environments, the best N was used. Figure 4 shows the percentage of members inserted into the population and at what position in the population they were inserted based on what part of the line they were generated with using the simplex hybrid with $N = 4$ on the BlueGene. The population is sorted from the best fit to the least, so the lower the position at which a member is inserted, the better its fitness with respect to the rest of the population. Figures 5 and 6 show the same information for BOINC and $N = 4$. To provide a measure of how far the population evolved while a member was being evaluated, these results are partitioned by how many other members were reported before the fitness of the current member was reported. The range of the probabilistic line search for the simplex was defined by limits $l_1 = -1.5$ and $l_2 = 1.5$ and the statistics are taken from five seperate searches.

On the BlueGene, the best insert rate and quality was from points around the centroid (generated between limits of 0.5 and -0.5). While inside of the worst point (1.0 to 0.5) had the highest insert rate, the quality of inserted members was rather low. Points near the reflection of the worst point through the centroid (-1.5 to -0.5) tended to have low insert rates, however when they were inserted they tended to be very fit. Points outside of the worst member (1.0 to 1.5) had the worst insert rate and the least fit. These results suggest that the probabilistic simplex search could be further optimized by restricting the range to limits $l_1 = -1.5$ and $l_2 = 0.5$, by eliminating the poorest performing range of 0.5 to 1.5.

BOINC showed similar results for quickly reported results (less than 200 members reported while the member was be-
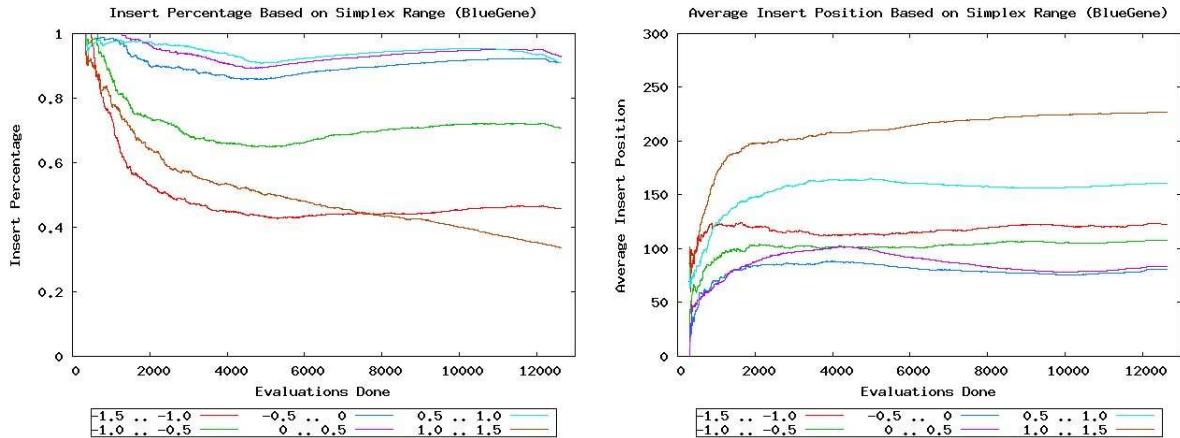
Figure 4: Average insert rate and insert position of members based on what part of the line calculated by the simplex hybrid they were generated on, for $N = 4$ using the BlueGene supercomputer. A lower insert position means the member is more fit than the rest of the population.

ing evaluated) with points generated near the centroid (-0.5 to 0.5) having the best fitness and insert rate (see Figures 5 and 6). One notable exception was that points generated on the inside of the worst point (0.5 to 1.0) had a notably lower insert rate and that points generated near the worst point (0.5 to 1.5) quickly degraded in terms of insert rate compared to other points. With over 1600 evaluations being reported during a members round trip time, not a single point generated past the worst point was inserted. Another point of interest is that while points generated near the reflection (-1.5 to -0.5) had lower insertion rates than those near the centroid (-0.5 to 0.5), as the report time increased, their average insert position stayed the same and eventually had better fitness than points generated near the centroid. As with the BlueGene, the results suggest that refining the limit on the probabilistic simplex operator to $l_1 = -1.5$ and $l_2 = 0.5$ would improve the convergence rates. Additionally, it appears that the result report time does have an effect on which part of the line used by the probabilistic simplex operator is better to draw new members from. An intelligent work scheduling mechanism could assign members generated near the reflection to processors with slower reporting times, and those generated near the centroid to processors with faster reporting times. Also, as the search progresses, there are fluctuations as to where the best points are generated from. An adaptive search could refine the limits to improve convergence rates. It is important to note that even the slowest processors retain their ability to evaluate members that are of benefit to the search, which is an important attribute for any algorithm running on massively distributed and heterogeneous environments.

5. DISCUSSION

This paper examines the use of asynchronous genetic search on the BOINC volunteer computing framework and compares it to synchronous continuously updated genetic search on a BlueGene supercomputer. The genetic search is used in a computationally intensive scientific application - optimizing the fit of an astronomical model of the Milky Way galaxy to observed stars. While the search converges in less

evaluations synchronously on the BlueGene, and the 1024 processor partition of the BlueGene provides a similar number of fitness evaluations per second as the approximately 1,000 user BOINC community, access to the BlueGene is limited and shared with other researchers. Alternatively, the BOINC project is dedicated to the astronomy project and operates at a fraction of the cost. Because of these factors, we argue that asynchronous genetic search on volunteer computing platforms is a valuable asset to scientific researchers doing computationally intensive scientific modeling and it is comparable with synchronous genetic search on supercomputing environments.

Two different crossover methods were compared: a double shot and a probabilistic simplex operator. The probabilistic simplex operator was shown to converge to the global minima faster than the double shot approach in all cases. Additionally, as more parents were used in the probabilistic simplex operator, the convergence rate increased. Analysis of the range from which the points were generated with this operator shows that there is definite room for improvement in multiple ways. For future work, increasing the number of parents to determine if and when this ceases to improve convergence is of interest. Also, the analysis suggests that bounds closer to the reflection and centroid points of the simplex would result in faster convergence or that an adapative approach that changes the bounds of the search based on how previously generated members have performed would improve performance. For BOINC, utilizing differing numbers of parents in the same search may also improve convergence, as at different times in the search progression, a smaller number of parents resulted in temporarily faster convergence. Yen et al. have also shown that using different points in the simplex for reflection, contraction and expansion may improve convergence [25] and using different points instead to generate the line for the probabilistic simplex operator may provide even better results.

Finally, as our BOINC community grows and more volunteered computers become accessible, the scalability of a single server may become a problem. In this case, extending the asynchronous genetic search strategy to utilize multiple

islands may be required. Also, modification of other search methods such as simulated annealing and particle swarm optimization for large scale asynchronous use may provide new insights and better convergence rates. As the number and heterogeneity of computers being used for computationally intense applications continues to increase, research into asynchronous searches of this nature will become increasingly important.

6. REFERENCES

[1] J. e. a. Adelman-McCarthy. The 6th Sloan Digital Sky Survey Data Release, http://www.sdss.org/dr6/, July 2007. ApJS, in press, arXiv/0707.3413.

[2] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.

[3] E. Alba and B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 9:126–142, April 2005.

[4] E. Alba and J. M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems*, 17:451–465, January 2001.

[5] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[6] D. P. Anderson, E. Korpela, and R. Walton. High-performance task distribution for volunteer computing. In *e-Science*, pages 196–203. IEEE Computer Society, 2005.

[7] J. Berntsson and M. Tang. A convergence model for asynchronous parallel genetic algorithms. In *IEEE Congress on Evolutionary Computation (CEC2003)*, volume 4, pages 2627–2634, December 2003.

[8] R. D. Blumofe and C. E. Leiserson. Scheduling Multithreaded Computations by Work Stealing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94)*, pages 356–368, Santa Fe, New Mexico, November 1994.

[9] E. Cantu-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998.

[10] R. Chelouah and P. Siarry. Genetic and nelder-mead algorithms hybridized for a more accurate global optimiation of continuous multiminima functions. *European Journal of Operational Research*, 148(2):335–348, july 2003.

[11] T. Desell, N. Cole, M. Magdon-Ismail, H. Newberg, B. Szymanski, and C. Varela. Distributed and generic maximum likelihood evaluation. In *3rd IEEE International Conference on e-Science and Grid Computing (eScience2007)*, pages 337–344, Bangalore, India, December 2007.

[12] B. Dorronsoro and E. Alba. A simple cellular genetic algorithm for continuous optimization. *IEEE Congress on Evolutionary Computation (CEC2006)*, pages 2838–2844, July 2006.

[13] G. Folino, A. Forestiero, and G. Spezzano. A JXTA based asynchronous peer-to-peer implementation of genetic programming. *Journal of Software*, 1:12–23, August 2006.

[14] H. Imade, R. Morishita, I. Ono, N. Ono, and M. Okamoto. A grid-oriented genetic algorithm framework for bioinformatics. *New Generation Computing: Grid Systems for Life Sciences*, 22:177–186, January 2004.

[15] V. Katari, S. C. Satapathy, J. Murthy, and P. P. Reddy. Hybridized improved genetic algorithm with variable length chromosome for image clustering. *IJCSNS International Journal of COmputer Science and Network Security*, 7(11):121–131, November 2007.

[16] A. Lewis and D. Abramson. An evolutionary programming algorithm for multi-objective optimisation. In *IEEE Congress on Evolutionary Computation (CEC2003)*, volume 3, pages 1926–1932, December 2003.

[17] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, and B.-S. Lee. Efficient hierarchical parallel genetic algorithms using grid computing. *Future Generation Computer Systems*, 23:658–670, May 2007.

[18] V. Pande et al. Atomistic protein folding simulations on the submillisecond timescale using worldwide distributed computing. *Biopolymers*, 68(1):91–109, 2002. Peter Kollman Memorial Issue.

[19] J. Purnell, M. Magdon-Ismail, and H. Newberg. A probabilistic approach to finding geometric objects in spatial datasets of the Milky Way. In *Proceedings of the 15th International Symposium on Methodoligies for Intelligent Systems (ISMIS 2005)*, pages 475–484, Saratoga Springs, NY, USA, May 2005. Springer.

[20] J.-M. Renders and H. Bersini. Hibridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In *IEEE World Congress on Computational Intelligence, First Conference on Evolutionary Computation*, volume 1, pages 312–317, June 1994.

[21] S. C. Satapathy, J. Murthy, P. Prasada, V. Katari, S. Malireddi, and V. S. Kollisetty. An efficient hybrid algorithm for data clustering using improved genetic algorithm and nelder mead simplex search. In *International Conference on Computational Intelligence and Multimedia Applications*, volume 1, pages 498–510, December 2007.

[22] G. Seront and H. Bersini. Simplex ga and hybrid methods. In *IEEE International Conference on Evolutionary Computation*, pages 845–848, May 1996.

[23] A. Sinha and D. E. Goldberg. A survey of hybrid genetic and evolutionary algorithms. Technical Report No. 2003004, Illinois Genetic Algorithms Laboratory (IlliGAL), 2003.

[24] L. Wei and M. Zhao. A niche hybrid genetic algorithm for global optimization of continuous multimodal functions. *Applied Mathematics and Computation*, 160(3):649–661, January 2005.

[25] J. Yen, J. C. Liao, B. Lee, and D. Randolph. A hybrid approach to modeling metabolic systems using a genetic algorithm and simplex method. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 29(2):173–191, April 1998.
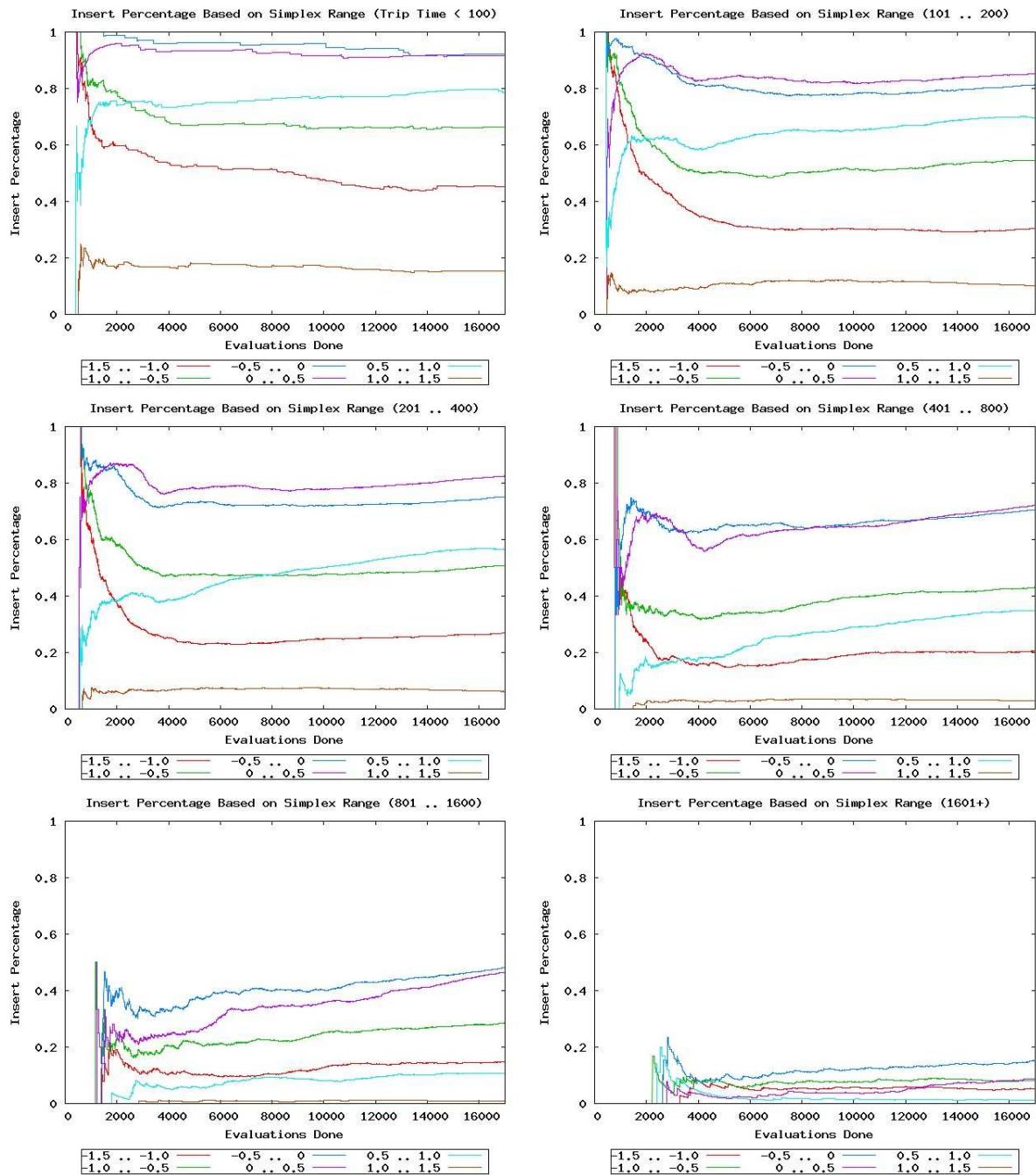
Figure 5: Average insert rate of members based on what part of the line calculated by the simplex hybrid they were generated on, for $N = 5$ using the BOINC framework. The results are partitioned by how many other members were reported while the used members were being generated (0..100, 101..200, 201..400, 401..800, 801..1600 and 1601+) to show the effects of asynchronicity and of a heterogeneous computing environment.
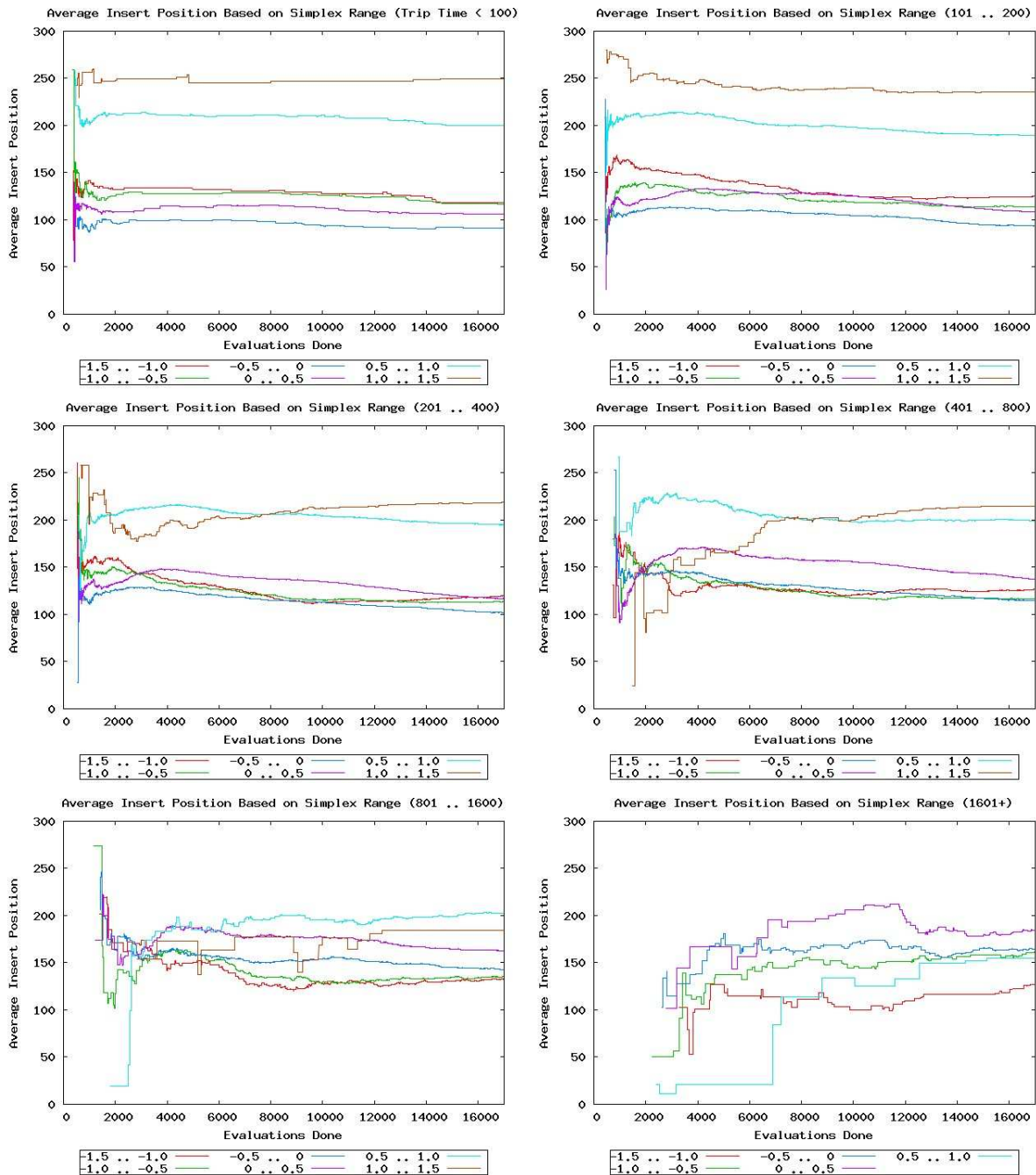
Figure 6: Average insert position of members based on what part of the line calculated by the simplex hybrid they were generated on, for $N = 4$ using the BOINC framework. A lower insert position means the member is more fit than the rest of the population. The results are partitioned by how many other members were reported while the used members were being generated (0..100, 101..200, 201..400, 401..800, 801..1600, 1601+) to show the effects of asynchronicity and of a heterogeneous computing environment.