Improving Evolutionary Algorithms with Scouting: High–Dimensional Problems

Konstantinos Bousmalis[1], Jeffrey O. Pfaffmann[2], and Gillian M. Hayes[3]

[1] School of Informatics, The University of Edinburgh, Edinburgh, UK
K.Bousmalis@sms.ed.ac.uk
[2] Department of Computer Science, Lafayette College, Easton, PA 18042, USA
pfaffmaj@cs.lafayette.edu
[3] Institute of Perception, Action and Behavior(IPAB), School of Informatics, The University of Edinburgh, Edinburgh, UK
gmh@inf.ed.ac.uk

Abstract. Evolutionary Algorithms (EAs) are common optimization techniques based on the concept of Darwinian evolution. During the search for the global optimum of a search space, a traditional EA will often become trapped in a local optimum. The Scouting-Inspired Evolutionary Algorithms (SEAs) are a recently–introduced family of EAs that use a cross–generational memory mechanism to overcome this problem and discover solutions of higher fitness. The merit of the SEAs has been established in previous work with a number of two and three-dimensional test cases and a variety of configurations. In this paper, we will present two approaches to using SEAs to solve high–dimensional problems. The first one involves the use of Locality Sensitive Hashing (LSH) for the repository of individuals, whereas the second approach entails the use of scouting–driven mutation at a certain rate, the Scouting Rate. We will show that an SEA significantly improves the equivalent simple EA configuration with higher–dimensional problems in an expeditious manner.

1 Introduction

1.1 Introduction to Evolutionary Algorithms

A simple Evolutionary Algorithm (EA) is an optimization technique that discovers satisfactory solutions to a given problem by evolving populations of candidate solutions over time. Natural evolution is simulated on this initial population by *(a)* assigning a measure of merit (fitness value) to each individual solution via a fitness function; *(b)* selecting a number of individuals for "parenthood" via a selection scheme; and *(c)* using these selected "parents" to create the next generation of solutions via a number of genetic operators, which usually include crossover (exchange of genes) and/or mutation (variation of one or more genes). This cycle of fitness assignment, selection and reproduction continues for a set number of generations or until a certain threshold of fitness has been reached.

1.2 Scouting-Inspired Evolutionary Algorithm: Previous Work

The essence of scouting is a cross-generational mechanism that considers past state space samples to optimize future sample generation for efficient state space exploration. [4, 10, 11, 14, 15] Scouting does this by estimating a current sample fitness, using a weighted-average of the k-Nearest Neighbors (k-NN) calculation of previous samples, and computing the difference to the actual fitness. This estimate-actual fitness difference is analogous to a "surprise-level", with a larger surprise generating a smaller variance in the generation of the next sample. A Scouting-Inspired Evolutionary Algorithm (SEA) relies on the repository of past samples to also direct the search for fitter solutions towards areas that are more "surprising," or areas that have been insufficiently explored. [4, 15] Thus, this behavior is an indirect solution to the problem of premature convergence, since the technique will migrate to different regions of the search space once a peak is effectively characterized in the past sample database.

SEA achieves this migration by slightly altering the original Scouting Algorithm. Initally, the fitness estimated based on the existing knowledge of the given search space. A number (k) of the geometrically nearest neighbors to the current individual is calculated and the weighted average of their fitness value is then the estimated fitness for the individual of interest, as outlined in (1):

$$fit_{estimate} = \frac{\sum_{i \in k} w_i \times fit_i}{\sum_{i \in k} w_i} \quad , \tag{1}$$

where $w_i = d_i^2$, d_i is the Euclidean distance between the query and individual i, and fit_i is the fitness of individual i. The selection of k is a design decision and set to $k = 8$ in previous and original work presented in this paper. The Euclidean distance is used because it is assumed the state space is not ill-scaled, and the Mahalanobis distance could be used in these cases. This estimate is then compared to the real fitness value of the individual of interest to decide how "surprising" the area sampled by the individual is.

Definition 1 *The surprise value of an individual s_{ind} is defined as the absolute difference of the estimated and actual fitness values.*

An SEA requires a mutation operator that introduces Gaussian variation to an individuals genome. This operator has the effect of increasing standard deviation when the surprise is low, and the inverse when surprise is high. In previous work [4] it has been shown that a good mapping between surprise (s_{ind}) and standard deviation (σ) is achieved by:

$$\sigma(s_{ind}) = \sigma_{max} - (s_{ind})^{\gamma} \times (\sigma_{max} - \sigma_{min}) \ , \ 0 < \gamma \leq 1 \ . \tag{2}$$

The boundaries of the standard deviation range, σ_{min} and σ_{max}, are a design decision and depend on the problem and its dimensionality. After analysis of the nature of the surprise values in [4], $\gamma = 0.01305$ was chosen for this parameter and we shall also use it throughout this paper.

1.3 SEA and the "Curse of Dimensionality"

The current scouting database implementation stores the datapoints it experiences in an unstructured manner and the k-Nearest Neighbor (k-NN) lookup is performed by sorting all points by their distance from the query datapoint. One simply has to consider an 100-dimensional problem with 100 individuals per generation for 30,000 generations; ignoring the space requirements, the current k-NN lookup (1) for the fitness estimate used in the surprise calculation makes the technique extremely slow —practically unusable.

The goal of the work presented here is to experiment with high–dimensional problems and show that an SEA configuration is significantly improving the performance of its equivalent EA. Taking advantage of recent work on k-NN optimization techniques, [7] we will present and use an alternative implementation of the scouting database that is designed for storing a large number of high–dimensional points and optimized for k-NN lookups. We will finally introduce the concept of *Scouting Rate* and present a few examples of how it could be useful to researchers who decide to use SEAs for their optimization problems, especially high–dimensional ones.

2 Approaches to an SEA for Higher–Dimensional Problems

The obvious approach to speeding up scouting–enhanced evolutionary algorithm configurations without trading for performance is the reimplementation of the SEA database. A number of alternative storing techniques were considered, but we focused on the state-of-the-art data storage structures for solving the k-Nearest Neighbor (k-NN) problem in high dimensions. [7] These were the Metric Trees [17], which are similar to the older Ball Trees, [13] and the Locality Sensitive Hashing (LSH). [3, 6, 9].

Another approach worth considering is using the scouting effect at a certain rate, which we call *Scouting Rate*.

2.1 Solutions to the k–Nearest Neighbor problem

Ball Trees and Metric Trees A Ball Tree or a Metric Tree is a binary tree where each node, a *ball*, represents a subspace of the n-dimensional Euclidean space bound by a hyper-sphere. The radius of the hyper-sphere is as large as required to contain the children nodes-balls. That means that the root node contains the entirety of the space covered by the datapoints in the structure. Sibling nodes are allowed to intersect and do not need to span the entire space. Ball and Metric Trees make k-NN search particularly simple and fast, since the datapoints are spatially organized and the search reduces to the problem of looking up the neighbors of a given query. The search is a depth-first one and significantly faster than the naïve linear search, like the one used by the current SEA implementation. [7]

Locality Sensitive Hashing (LSH) Locality Sensitive Hashing (LSH) is not a structure for solving the exact k-NN problem, but it is instead designed and considered the best approach for queries that require the approximate nearest neighbors within a neighborhood radius and a certain probability. [7] This problem is referred to as the *(R, 1-δ)-near neighbor problem*, where R is the neighborhood radius and δ the probability with which an actual neighbor is *not* reported.

The way LSH achieves a fast solution to the *(R, 1-δ)-near neighbor problem* is by hashing geometrically-close datapoints to the same "bucket". [6] The "buckets" are aligned with a prespecified resolution along the axes. The method's speed has been proved both theoretically and in practice. [3, 9] The details of this technique are complex and beyond the scope of this paper. The interested readder is referred to the concise explanation in [2].

The LSH approach and the official implementation [2] allowed for results that satisfy the goals of this research. Thus, the underlying structure provided by the Metric and Ball Trees is not required for the current paper.

2.2 Scouting Rate

Definition 2 *Scouting Rate, denoted as p_s, is the probability with which the standard deviation used during scouting-driven mutation is altered based on the fitness of the nearest neighbors of the individual of interest.*

The rationale behind using a scouting rate is that it is possible to avoid entrapment in local optima by only using scouting-driven mutation on a small portion of the population. Moreover, it is certain that using a scouting rate significantly lowers the number of k-Nearest Neighbor calculations that have to be performed during an experiment. These calculations, and not storing the individuals themselves, is what makes all traditional k-NN techniques slower and almost unusable in higher dimensions.

In this paper, we will show that a scouting rate as low as 0.5 still allows an SEA to significantly improve the equivalent EA, while cutting the cost of k-Nearest Neighbor lookups to half.

3 Implementation

3.1 Test Cases and the TCG-2 Package

As with previous work [4, 15], the Test–Case Generating Package TCG-2 [16] will be used to create a variety of fitness landscapes in order to assess how effective SEAs are in higher–dimensional domains. Previous work and especially [15] fully outline the merits of TCG-2 and the reasons for using it when experimenting with SEAs. In short, TCG-2 is a very configurable C++ package and can create a vast array of non-linear programming (NLP) tasks with different levels of complexity, modality and difficulty for an EA. In this paper we experiment with test–cases of 10, 50 and 150 peaks, peak width of σ_{peak}=0.1 and 0.2, and for

6, 8, 10 and 15 dimensions. The rest of the required parameters for TCG-2 are fixed and shown in Table 1.

Table 1. TCG-2 parameters for the high-dimensional experiments.

Number of feasible components	(m) : 5
Search space feasibility	(ρ) : 0.5
Search space complexity	(c) : 0
Active constraints at global optimum	(a) : 0
Peak decay	(α) : 0.1
Component minimum distance	(d) : 0.01
Penalty	(W) : 10

3.2 The Evolutionary Algorithm Framework

The Evolutionary Algorithm framework was implemented in C/C++ and follows the guidelines provided in the first two papers on SEAs [4, 15], with the necessary changes for the goals of this research. The EA/SEA configuration used here is the EAC/SEAC, as defined in [4]. This configuration uses Roulette Wheel selection, random deletion, single–point crossover at a rate of 0.5, and random (EAC) or scouting–driven (SEAC) mutation at a rate of 0.5. Minimum standard deviation for scouting–driven mutation σ_{min} and the fixed standard deviation for the EAC σ_{EA} are both set to $\sigma_{min} = \sigma_{EA} = 0.0107$, whereas the maximum one is $\sigma_{max} = 0.4$ for fitness landscapes with narrow peaks ($\sigma_{peak} = 0.1$), and $\sigma_{max} = 0.9$ for landscapes with wide peaks ($\sigma_{peaks} = 0.2$).

The evolutionary process loops for a set number of generations—30,000 for all experiments presented here, and the population size is 20 individuals. Finally, all experiments were run 75 times, using 25 different seeds generated with dice and three different random–number generators. The latter are provided by the GNU Scientific Library [5] and are the "Mersenne Twister" [12] and the two "ranlux" algorithms [8].

3.3 Locality Sensitive Hashing and the E^{2}LSH Package

As explained earlier, the reimplementation of the scouting database is essential to this work and preliminary investigation showed that the Locality Sensitive Hashing (LSH) approach is the most suitable one for our needs. The official LSH implementation is the E^{2}LSH (Exact Euclidean LSH) 0.1 package by Alexandr Andoni and Piotr Indyk. [2] It is a C/C++ package and was easily incorporated into the existing SEA project.

Running experiments with E^{2}LSH was much faster compared to the naïve scouting database without affecting the performance of the SEA. Table 2 shows the averages over 10 identical runs of each sample problem with the naïve scouting implementation, and the ones with E^{2}LSH. The E^{2}LSH parameters used for

these timing experiments are the ones in the column titled "Higher-Dim. TCG-2" in Table 3 and it is very likely that there exist other such parameters that make the SEA perform even faster for the problems presented.

Table 2. Naïve vs. E^{2}LSH implementation times. All experiments were run with a population size of 10 individuals.

TCG-2 problem	Naïve	E^{2}LSH
2D, 1000gens	5.8032sec ± 0.1085sec	19.3287sec ± 0.1067sec
3D, 1000gens	7.4905sec ± 0.63sec	8.0773sec ± 0.5396sec
3D, 5000gens	1min 39.4522sec ± 7.4372sec	43.5159sec ± 2.3219sec
10D, 1000gens	46.6837sec ± 7.6753sec	15.4688sec ± 1.2717sec
20D, 1000gens	61.8342sec ± 12.3005sec	17.4135sec ± 0.2978sec
40D, 1000gens	Unable to cope (Seg. Fault)	25.1240sec ± 1.0382sec
100D, 1000gens	Unable to cope (Seg. Fault)	54.3983sec ± 0.5909sec
300D, 1000gens	Unable to cope (Seg. Fault)	51.0080sec ± 1.3616sec

Table 3. The E^{2}LSH parameters used for the experiments presented in this paper.

Parameter	Higher-Dim. TCG-2	15-Dim. TCG-2
k	20	6
m	35	7
L	595	21
w	4	4

The speed improvement of the scouting database with E^{2}LSH was significant when the problem dimensionality was higher than two. The time duration for the three-dimensional example for 1000 generations was almost identical for the two techniques, but it is obvious that E^{2}LSH deals with an increasing number of points in a much more effective manner than the naïve implementation. Moreover, the E^{2}LSH implementation performed much better in the ten and twenty-dimensional examples, cutting down the time to more than one third. Finally and most importantly, the new database implementation performed better in the 300-dimensional example than the naïve one performed in the twenty-dimensional example.

It is important however to note that the times presented in Table 2 can be affected by the local memory resources available, the parameters for the E^{2}LSH package, the way the search is performed —the maximum standard deviation scouting mutation uses, for example— and even the task at hand.

The package includes a parameter calculator, but it requires a sample of datapoints from the problem search space. The creators advised that the parameter calculator should be invoked after a certain number of points has been experienced, at which point scouting could begin playing a role. [1] Our hope was

that we could avoid this step, given speed considerations, and we attempted to use the parameter calculator for a number of point samples to get an empirical understanding of the parameters and adapt them accordingly manually. Table 3 outlines the parameters used for the experiments presented in this paper.

4 Results and Discussion

Our goal in this paper is to show that an SEA configuration improves its equivalent EA in higher–dimensional problem domains. We therefore ran the EAC and the SEAC on TCG-2 test–cases of 6, 8, 10 and 15 dimensions. The latter were varied in the number of peaks —10, 50, 100 and 150— and their width —0.1 and 0.2.— We ran these with p_s=1 and with p_s=0.5 to show that even a scouting rate of 0.5 is enough to significantly improve the equivalent EA, by performing only 50% of the k-NN queries.

During experimentation with these high–dimensional problems, it became apparent that every problem has an optimal σ_{max}. A higher σ_{max} made the SEAC perform worse when dealing with narrower peaks and higher dimensions. The reason for that is the fact that changing all genes by the same amount creates individuals at a lower resolution as the number of dimensions increase — the created individuals have a larger distance among them by default. Assuming the highest value in the range given by each σ and using the fact that the range is sixfold the standard deviation (σ), the distance among individuals as a function of σ and the problem dimensionality is given by the following:

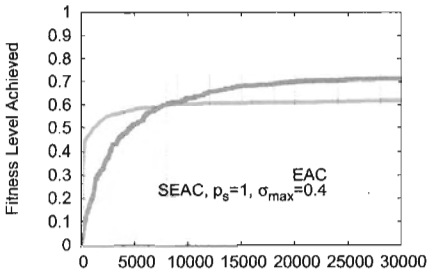
$$d(\sigma, n) = \sqrt{\left(\frac{6 \times \sigma}{2}\right)^2 \times n} \tag{3}$$

This makes it harder to find only a few narrow peaks in a large zero–fitness plateau when using a large σ_{max}. We therefore picked to use σ_{max}=0.4 for the TCG-2 cases with σ_{peak}=0.1 and σ_{max}=0.9 for those with σ_{peak}=0.2.

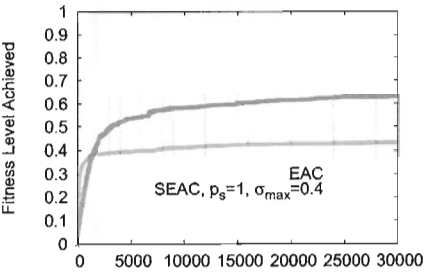
All the results obtained showed an improvement of performance when scouting–driven mutation was in use. Figures 1, 2, 3 and 4 show the performance of the SEAC with $p_s = 1$ against the equivalent EAC for only the 15–dimensional cases, due to lack of space. As one can clearly see, in every case the SEAC improves significantly after a certain generation continuing to improve up to and beyond the limit of 30,000 generations, whereas the EAC reaches a plateau much earlier, in the ca. $2{,}000^{th}$ generation for most cases presented here.

In the cases of 10 and 50 peaks, the slope of the SEAC curve is steeper and still increases rapidly at the end of the evolutionary process, whereas in the cases of 100 and 150 peaks, the SEAC has already reached a plateau at the same point. It is important to note that in the worst case the SEAC always performs significantly better than the worst case of the EAC and sometimes better than the average case of the latter. (see Fig. 3(b) for an example)

The results obtained also showed that only a small number of scouting–driven mutations are enough to get a population out of local optima and improve the

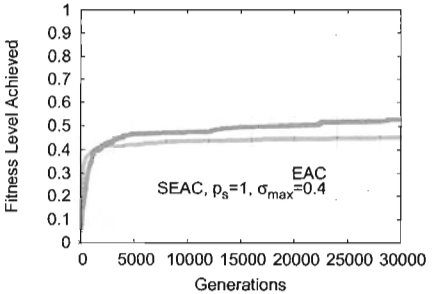


(a) 10 peaks, σ_{peak}=0.1

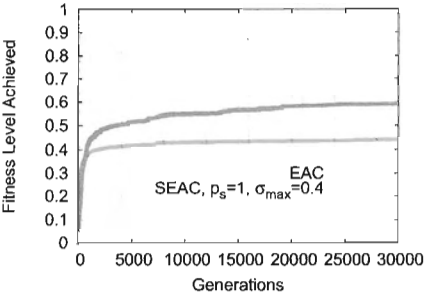


(b) 50 peaks, σ_{peak}=0.1

Fig. 1. Fitness level achieved per generation by EAC vs SEAC, for the 15–dimensional test cases with $\sigma_{peak} = 0.1$ for 10 and 50 peaks.

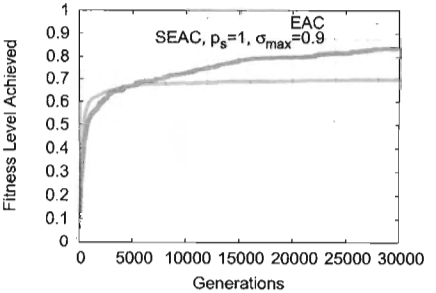


(a) 100 peaks, σ_{peak}=0.1

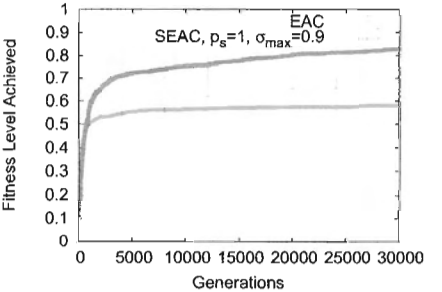


(b) 150 peaks, σ_{peak}=0.1

Fig. 2. Fitness level achieved per generation by EAC vs SEAC, for the 15–dimensional test cases with $\sigma_{peak} = 0.1$ for 100 and 150 peaks.

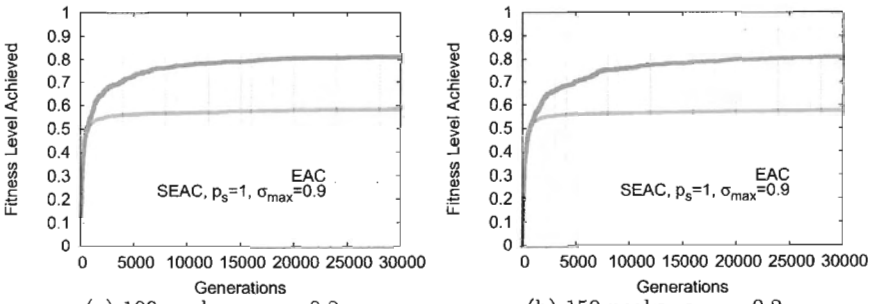


(a) 10 peaks, σ_{peak}=0.2



(b) 50 peaks, σ_{peak}=0.2

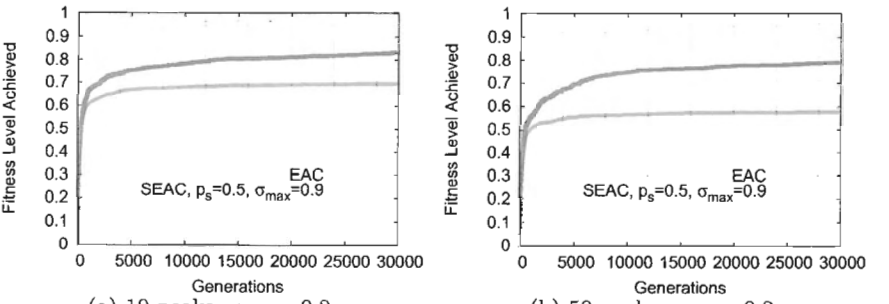
Fig. 3. Fitness level achieved per generation by EAC vs SEAC, for the 15–dimensional test cases with $\sigma_{peak} = 0.2$ for 10 and 50 peaks.



(a) 100 peaks, σ_{peak}=0.2 (b) 150 peaks, σ_{peak}=0.2

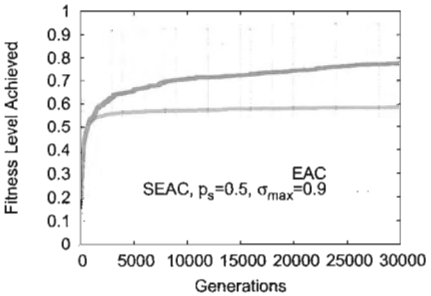
Fig. 4. Fitness level achieved per generation by EAC vs SEAC, for the 15–dimensional test cases with $\sigma_{peak} = 0.2$ for 100 and 150 peaks.

relevant typical EA configuration, as shown in Figs. 5 and 6 for four of the problems we experimented with. Experiments with even lower scouting rate — p_s=0.02 and 0.2— showed that even when scouting–driven mutation is applied on only 2% of the population, the SEAC still exhibited a significant improvement over the EAC in most cases. We would generally suggest the use of a low scouting rate if speed is important, and a high rate if the highest possible fitness level is crucial, or if fewer passes are required, due to the cost of the fitness function for example.

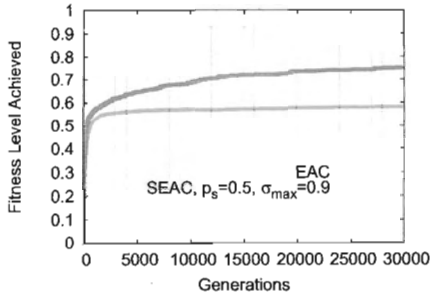


(a) 10 peaks, σ_{peak}=0.2 (b) 50 peaks, σ_{peak}=0.2

Fig. 5. Fitness level achieved per generation by EAC vs SEAC with $p_s = 0.5$, for the 15–dimensional test cases of $\sigma_{peak} = 0.2$ with 10 and 50 peaks.



(a) 100 peaks, σ_{peak}=0.2



(b) 150 peaks, σ_{peak}=0.2

Fig. 6. Fitness level achieved per generation by EAC vs SEAC with $p_s = 0.5$, for the 15–dimensional test cases of $\sigma_{peak} = 0.2$ with 100 and 150 peaks.

5 Conclusion-Future Work

In conclusion, we have showed that an SEA performs better than the EA it improves even in higher–dimensional cases, provided a good choice of σ_{max}. This has become possible with the implementation of a new scouting database, based on the Locality Sensitive Hashing (LSH) and the use of the E^{2}LSH package, which has accelerated the SEA by dramatically reducing the k-NN lookup time. We have introduced the concept of a scouting rate and showed that using scouting–driven mutation on only a small percentage of the individuals is sufficient to push the entire population away from local optima and towards other, fitter solutions.

The obvious next step in this research is the incorporation of the E^{2}LSH parameter calculator. One suggestion on how to achieve this efficiently is to calculate the parameters only during the first pass of an SEA on a problem and use them for the rest of the passes. It is also important for future work to include experimentation with a variety of σ_{max} and scouting rate values, in order to fully understand the way these affect the performance of the technique. We speculate that scouting rate could be smaller as the population size increases, and future work could also experiment with higher population sizes and deduce accordingly.

An alternative avenue of exploration could include turning the scouting effect off for a number of generations, approximately the number of generations required by the EAC to reach a plateau on average. During that time the SEAC would operate with $p_s = 0$ and $\sigma_{min} = \sigma_{max}$ and attempt to populate the experience database fast so that the SEAC would have more information and therefore be more effective when scouting does gets enabled.

References

[1] A. Andoni. Direct, unpublished correspondence, July 2007.

[2] A. Andoni and P. Indyk. *E^2LSH 0.1 User Manual.* MIT, June 21 2005.

[3] A. Andoni and P. Indyk. Near-optimal hashing algorithms for near neighbor problem in high dimensions. In *Proceedings of the Symposium on Foundations of Computer Science–FOCS '06*, pages 459–468, 2006.

[4] Konstantinos Bousmalis, Gillian M. Hayes, and Jeffrey O. Pfaffmann. Improving evolutionary algorithms with scouting. In *Main Proceedings of the 13^{th} Portuguese Conference on Artificial Intelligence—EPIA 2007*, Lecture Notes in Computer Science. Springer–Verlag, December 3-7 2007.

[5] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *GNU Scientific Library Reference Manual.* Network Theory Ltd., Bristol, UK, 2003.

[6] A. Gionis, P. Indyk, and R Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, September 07-10 1999.

[7] T. Liu, W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *Proceedings of Neural Information Processing Systems*, 2004.

[8] M. Lüscher. A portable high-quality random number generator for lattice field theory calculations. In *Computer Physics Communications*, volume 79, pages 1000–1110, 1994.

[9] Datar M., P. Indyk, N. Immorlica, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Symposium on Computational Geometry*, 2004.

[10] N. Matsumaru, S. Colombano, and K.P. Zauner. Scouting enzyme behavior. In D.B. Fogel, M.A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, editors, *2002 World Congress on Computational Intelligence*, pages 19–24, Honolulu, Hawaii, 12-17 May 2002. IEEE, Piscataway, NJ.

[11] N. Matsumaru, F. Centler, K.P. Zuner, and P. Dittrich. Self-adaptive-scouting autonomous experimentation for systems biology. In *EvoWorkshops 2004, LNCS 3005*, pages 52–62, 2004.

[12] M. Matsumoto and T. Nishimura. Mersenne twister: A 6234-dimensionally equidistributed uniform pseudo-random number generator. In *ACM Transactions on Modeling and Computer Simulation*, volume 8, pages 3–30, 1998.

[13] S.M. Omohundro. *Five balltree construction algrotihms.* International Computer Science Institute, Berkeley, CA, November 20 1989.

[14] J.O. Pfaffmann and K.P. Zauner. Scouting context-sensitive components. In D. Keymeulen, A. Stoica, J. Lohn, and R.S. Zebulum, editors, *The Third NASA/DoD Workshop on Evolvable Hardware-EH2001*, pages 14–20. IEEE Computer Society, 12-14 July 2001.

[15] J.O Pfaffmann, K. Bousmalis, and S. Colombano. A scouting-inspired evolutionary algorithm. In *Proceedings of the 2004 Congress on Evolutionary Computation–CEC2004*, volume 2, pages 1706–1712, Portland, OR, 16-19 June 2004.

[16] M. Schmidt and Z. Michalewicz. Test-case generator tcg-2 for nonlinear parameter optimization. In K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J.M.

Guervos, and H.-P. Schwefel, editors, *Proceedings of the 6th International Conference in Parallel Problem Solving from Nature - PPSN VI*, volume 1917 of Lecture Notes in Computer Science, pages 539–548, Paris, France, 18-20 September 2000. Springer-Verlag.

[17] J.K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. In *Information Processing Letters*, number 40, pages 175–179, November 1991.

Acknowledgments. The authors would like to thank Alexandr Andoni and Piotr Indyk for the E^2LSH package and valuable advice regarding its use.