A Steady-State Genetic Algorithm With Resampling for Noisy Inventory Control*

Steven Prestwich[1], S. Armagan Tarim[2], Roberto Rossi[1] and Brahim Hnich[3]
[1] Cork Constraint Computation Centre, University College, Cork, Ireland
s.prestwich@cs.ucc.ie, r.rossi@4c.ucc.ie
[2] Department of Management, Hacettepe University, Turkey
armagan.tarim@hacettepe.edu.tr
[3] Faculty of Computer Science, Izmir University of Economics, Turkey
brahim.hnich@ieu.edu.tr

Abstract. Noisy fitness functions occur in many practical applications of evolutionary computation. A standard technique for solving these problems is fitness resampling but this may be inefficient or need a large population, and combined with elitism it may overvalue chromosomes or reduce genetic diversity. We describe a simple new resampling technique called Greedy Average Sampling for steady-state genetic algorithms such as GENITOR. It requires an extra runtime parameter to be tuned, but does not need a large population or assumptions on noise distributions. In experiments on a well-known Inventory Control problem it performed a large number of samples on the best chromosomes yet only a small number on average, and was more effective than four other tested techniques.

1 Introduction

In many real-world applications of Genetic Algorithms (GAs) and other Evolutionary Computation algorithms, the fitness function is *noisy*: that is, the fitness of a chromosome cannot be computed directly but must be averaged over a number of samples. Examples include the learning of randomised games such as Backgammon, human-computer interaction, and simulation problems for which we wish to evolve a robust plan. The standard deviation of the sample mean of a random variable with standard deviation σ is $\sigma/\sqrt{n}$ where n is the number of samples, so a large number of samples may be needed for very noisy fitness functions.

Several techniques for handling fitness noise in EAs are surveyed in [4, 13]: the use of sampling to obtain an average fitness reduces noise; increasing the population size makes it harder for an unfit chromosome to displace a fitter one

* S. A. Tarim and B. Hnich are supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. SOBAG-108K027. R. Rossi is supported by Science Foundation Ireland under Grant No. 03/CE3/I405 as part of the Centre for Telecommunications Value-Chain-Driven Research (CTVR) and Grant No. 05/IN/I886.

(a point also made by [10]) and can be viewed as a form of implicit averaging; and *rescaled mutation* samples distant points in the search space then moves a small distance toward them. [5] propose regression to estimate the fitness of neighbouring chromosomes. [1] vary sample rates across both chromosomes and generations in a generational GA. [18] record fitness levels in a search history, and use a stochastic model of fitness levels to locate new points in the search space. [3] use a threshold selection heuristic for accepting chromosomes. [17] adapt the sampling rate to different regions of the search space, a technique they call *dynamic resampling*. [19] use a Bayesian approach to sampling called Optimal Computing Budget Allocation, which assumes normally distributed noise.

A popular approach is to use a *Noisy Genetic Algorithm* (NGA) which computes the fitness of each chromosome by averaging over a number of samples [9, 11, 14, 15]. Following [1] we shall refer to this as *static sampling*, and refer to this algorithm as NGAs. NGAs wastes considerable time evaluating unpromising chromosomes, but it can be improved by linearly increasing the number of samples with search time, starting from a low value [21, 27]. We shall refer to this as *incremental sampling* and the resulting algorithm as NGAi. However, though NGAs and NGAi have been used to solve real problems, they may not be the most efficient approach. It is pointed out in [22] that a reduction in noise is not necessary for *every* chromosome, only for the *best* ones. Of course, this entails discovering which are the best chromosomes without performing a large number of samples, but poor chromosomes might become apparent after just a few samples.

An alternative technique is to *resample* chromosome fitness: that is, some chromosomes are allowed to survive for more than one generation, and their fitness is periodically recomputed to refine the estimate. Various heuristics may be used to decide when to discard a chromosome. [22] experiments with averaging over a small number of samples, and guiding resampling by a statistical test which assumes Gaussian noise but is considered to be robust under non-Gaussian noise. [12] uses the standard deviation of the fitness to correct for its noise, again under assumptions on noise distribution. Resampling and the common heuristic of *elitism* do not always combine well. [6] show that, with an elitist GA, the probabilistic method of [12] is inferior to a resampling approach. [2] show that, in Evolutionary Strategies that allow fitness values to survive for more than one generation, failure to resample can lead to systematic overvaluation of chromosomes. [8] found that, when applying co-evolutionary learning to the noisy task of learning how to play Backgammon, more sampling can have a bad effect on the learning besides incurring overhead. It causes less fit chromosomes to be pruned more quickly which reduces genetic diversity too drastically, especially with small populations. Despite these drawbacks, resampling and elitism have been successfully combined. [25] describe an extension of the Simple (generational) GA that maintains a list of the fittest solutions found so far, while increasing the number of samples as search proceeds as in NGAi; they also increase the population size during search.

Another successful resampling elitist GA is the *Kalman-extended Genetic Algorithm* (KGA) [23], designed for problems whose fitness is both noisy and nonstationary. It adapts its sampling rate for each chromosome individually, based on techniques from Kalman filtering. Removing the nonstationary aspects of KGA yields a steady-state algorithm that evaluates the fitness of each new chromosome just once before adding it to the population, then replaces the least-fit population member by the new chromosome. Alternate iterations are devoted to resampling chromosomes that are already in the population. The current fitness estimate of a chromosome is the mean over all its samples. In KGA a chromosome is selected for resampling according to its current fitness estimate and how many times it has already been sampled (which is a measure of the fitness uncertainty): choose the chromosome with fewest samples, among those whose fitness estimates are greater than the population fitness mean minus the population fitness standard deviation. The intuition behind this approach is that unfit chromosomes with high fitness estimate based on only a few samples will be resampled, and their low fitness will become apparent. We shall refer to this as *Kalman sampling*.

In this paper we investigate resampling strategies for the steady-state (therefore elitist) GENITOR algorithm [26]. Our aim is to find a simple resampling strategy that can be used with a steady-state GA, does not assume any noise properties, does not require a large population, resamples fit chromosomes many times to avoid overvaluation, yet on average uses only a few samples per chromosome. We find it necessary to introduce a new runtime parameter that requires manual tuning, but this might be automated in future work. We demonstrate our technique on a well-known problem from Inventory Control. Section 2 describes our algorithm, Section 3 describes the problem we attempt to solve, Section 4 presents experimental results, and Section 5 concludes the paper.

2 The algorithm

We use a single GA in our experiments: a basic version of GENITOR [26] without refinements such as a gene to determine crossover probability. GENITOR is a steady-state GA that, at each iteration, selects two parent chromosomes, breeds a single offspring by (optional) crossover followed by mutation, evaluates it, and uses it to replace the least-fit member of the population. We use random parent selection, and standard uniform crossover applied with a crossover probability 0.5: if it is not applied then a single parent is selected and mutated. In our problem (described below) each gene can take any of 100 integer values, plus a special value denoted by NULL. Because of the special nature of the NULL value we select it with probability 0.5, otherwise randomly select one of the 100 integer values. Mutation is applied to a chromosome once with probability 0.5, twice with probability 0.25, three times with probability 0.125, and so on. A small population of size 30 is used. We assume that at least U samples are required to obtain a sufficiently reliable fitness estimate, and in experiments we will use the large value $U = 1000$. Thus we face the challenge of sampling

effectively without incurring the drawbacks described above: inefficiency, lack of genetic diversity, or overvaluation, while using only a small population.

This is our basic GA but we have yet to specify a sampling strategy to cope with fitness noise. We will compare five resampling strategies, three of which are well-known: static sampling (as in NGAs) in which we take U samples for each chromosome, incremental sampling (as in NGAs) in which we take a variable number of samples per chromosome that linearly increases from 1 to U during the GA execution, and Kalman sampling (as in KGA). The other two strategies are new.

Our first new strategy tries to combine the rapid convergence of Kalman sampling with the reliability of static sampling. It applies Kalman sampling but with a number $S \geq 1$ of samples to initialise and resample chromosomes, with the best value of S to be determined by experiment. We shall refer to this as *Kalman averaged sampling* and our GA with this sampling scheme as KASGA. It is inspired by a note in [1] stating that if the fitness variance in the population is small compared to the noise variance then a GA will make no progress, and it becomes necessary to increase the sample rate. It is also inspired by the use of a small number of samples for evolutionary algorithms in [22].

Our second new strategy also takes S samples each time a chromosome is selected for (re)sampling, but it resamples the chromosome with highest fitness, ignoring chromosomes that already have U samples. Note that if $S < U$ then there is always at least one chromosome with fewer than U samples: the most recently created chromosome, which only has S samples. Note also that we normally choose S to be a divisor of U to avoid unnecessary resampling, but this is not strictly required. We shall call this scheme *greedy averaged sampling* because it greedily resamples the most promising chromosome, based on current fitness estimates. Combining this with the GA we obtain a new algorithm we shall call the Greedy Average Sample GA (GASGA). This is our main contribution and it is summarised in Figure 1.

```
GASGA(S, P, U)
  create population of size P
  evaluate population using S samples
  while not(termination condition)
    select two parents
    breed one offspring O
    evaluate O using S samples
    replace least-fit chromosome by O
    select fittest chromosome F with #samples < U
    re-evaluate F using S samples
  output fittest chromosome
```

Fig. 1. GASGA pseudo-code

3 An inventory control problem with uncertainty

The problem we consider is as follows. Given a planning horizon of N periods and a demand for each period $t \in \{1, \ldots, N\}$, which is a random variable with a given probability density function; we assume that these distributions are normal, though this is not required by our GA. Demands occur instantaneously at the beginning of each time period and are *non-stationary* (can vary from period to period), and demands in different periods are independent. A fixed delivery cost a is incurred for each order, a linear holding cost h is incurred for each product unit carried in stock from one period to the next, and a linear stockout cost s is incurred for each period in which the net inventory is negative (it is not possible to sell back excess items to the vendor at the end of a period). The aim is to find a replenishment plan that minimizes the expected total cost over the planning horizon.

Different inventory control policies can be adopted to cope with this and other problems. A policy states the rules used to decide when orders are to be placed and how to compute the replenishment lot-size for each order. (The term *policy* here refers to the *form* of the plan, whereas in some fields such as Artificial Intelligence a policy refers to an *actual* plan. We use the term in both senses, and the meaning should be clear from the context.) One possibility is the *replenishment cycle policy* (R, S) [20]. With non-stationary demands this policy takes the form (R^n, S^n) where R^n denotes the length of the n^{th} replenishment cycle and S^n the order-up-to-level for replenishment. In this policy a wait-and-see strategy is adopted, under which the actual order quantity for replenishment cycle n is determined only after the demand in former periods has been realized. The order quantity is computed as the amount of stock required to raise the closing inventory level of replenishment cycle $n-1$ up to level S^n. To provide a solution we must populate both the sets R^n and S^n for $n = \{1, \ldots, N\}$. The (R, S) policy yields plans of higher cost than optimal but has been formulated to reduce *nervousness* in inventory control, and is more often used in practice.

There are more efficient algorithms which are guaranteed to yield optimal policies (under reasonable simplifying assumptions) so a GA would not be applied to precisely this problem in practice. However, if we complicate the problem in simple but realistic ways, for example by adding order capacity constraints or dropping the assumption of independent demands, these efficient algorithms become unusable. In contrast, a GA can be used almost without modification. Thus the problem is useful as a representative of a family of more complex problems.

The replenishment cycle policy can be modelled as follows. Each chromosome represents a single policy, each gene corresponds to a period n, an allele specifies the order-up-to level or the lack of an order (denoted here by the special value NULL) for that period, and a chromosome's fitness is the inverse of the total cost incurred by the policy that it represents. For our experiments we allow 100 different order-up-to levels, linearly spaced in the range 1–300. Thus each gene has 101 alleles. These parameters were chosen as suitable for the instances we tested.

4 Experiments

We obtained results using several problem parameter settings, and in each case found the same relationships between the algorithms. For this reason, and because of limited space, we present results for only one instance: 100 periods, stationary demands with mean 50 and standard deviation 10 in all periods, and cost parameters $h = 1$, $a = 400$ and $s = 10$. Problems with 100 periods are very hard: none of the methods we test can find the optimal policy within several hours (nor did attempts using Mixed Integer Programming and Reinforcement Learning algorithms). The optimal policy has an expected total cost of 19,561 with replenishment every 4 periods (starting from the first period) and order-up-to levels of 205 deduced from the cyclic nature of the problem (which is not exploited by the algorithms we test).

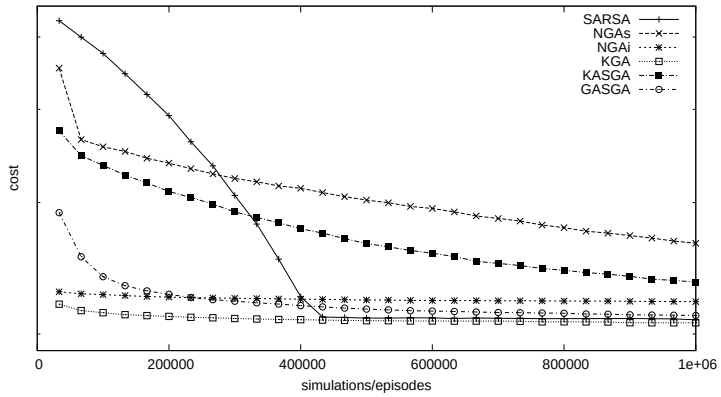
We will compare several GAs using three metrics: the *fitness* of the selected chromosome, the *reliability* of the selected chromosome measured by the number of samples used to compute the fitness, and the *wastefulness* of the GA measured by the number of samples used to estimate the fitness of discarded chromosomes. Almost every chromosome is discarded at some point during search, so the wastefulness is an approximation to the average number of samples used per chromosome. Ideally we aim for a GA with high fitness and reliability, but low wastefulness. In our experiments we aim for a reliability of $U = 1000$. The results are shown in Figure 2.

The fitness graph also shows results for the SARSA(λ) Reinforcement Learning algorithm [24] for comparison, as the problem can be modelled as an episodic Partially Observable Markov Decision Process in which a *state* is the period, an *action* is either the choice of an order-up-to level or the lack of an order (NULL) in a period, and a *reward* (undiscounted) is minus the total cost incurred in a period. We use an ϵ-greedy heuristic, varying ϵ inversely with time as recommended in [24], and tuning the α, λ parameters by the common method of hill-climbing in parameter space. All state-action values were initialised to 0, as the use of optimistic initial values encourages early exploration [24].

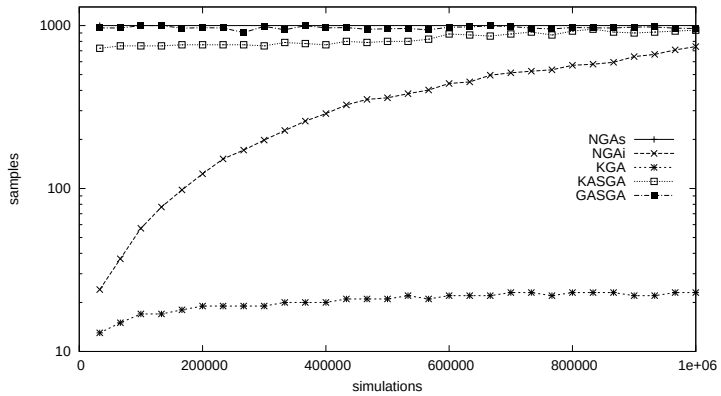
Because there is a range of Pareto-optimal solutions among the chromosomes of a GA, varying from high fitness based on few samples to low fitness based on many samples, we have a problem: how should different GAs be compared? We are interested in fit solutions based on many samples, so for each GA we shall select the chromosome with the greatest value of samples/cost. The results are as follows.

The graphs show that NGAs has high reliability, but it converges quite slowly and has high wastefulness as it uses exactly 1000 samples for every chromosome. NGAi has much better fitness than NGAs. It reaches this fitness rapidly but then make little further progress, perhaps because of its increasing wastefulness. However, it achieves NGAs's reliability by the end of the run, and only matches its wastefulness by the end of the run. Note that the reliability does not quite reach 1000 samples: there is a delay between (i) increasing the number of samples to a given number, and (ii) obtaining a chromosome whose fitness is both high and based on that number of samples. This delay would not occur in a

fitness:



reliability:



wastefulness:

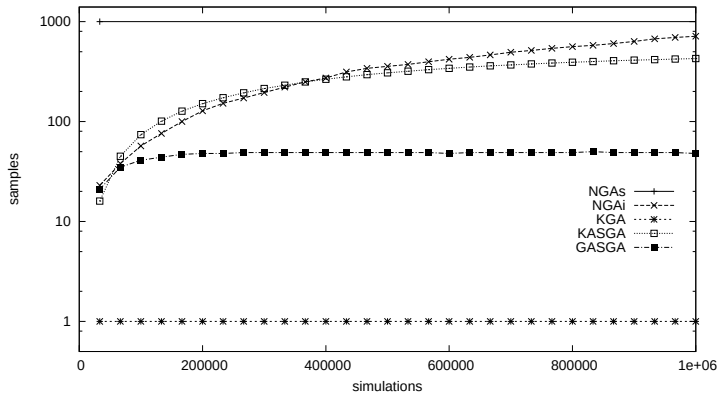


Fig. 2. Experimental results

generational GA, in which no chromosome survives to the next generation. We should perhaps use a generational GA to evaluate incremental sampling, as this was the form of GA used in the Noisy GA work, but in this paper our aim is to compare several sampling techniques on the same (steady-state) GA. However, a generational GA will presumably exhibit similar wastefulness.

KGA has excellent fitness but very low reliability. Though KGA has given good results on other problems, here no chromosome survives long enough to achieve a sufficient number of samples. This is caused by the high fitness noise in our problem: as chromosomes are resampled their estimated fitnesses fluctuate significantly, and over many iterations the fittest chromosome is not much more likely to survive than any other. Our problem is very noisy, with the fitness standard deviation not much less than the mean, and KGA seems unsuitable for such problems. KASGA is a marked improvement over KGA. Increasing S until the reliability is approximately 1000 samples, we reach a value $S = 250$. The graphs show that KASGA has better fitness than NGAs but no other algorithm, probably because of its fairly high wastefulness (approximately 400 samples per chromosome). But it does have high reliability, making it more usable than KGA.

GASGA outperforms KASGA and the other algorithms. Again increasing S until reliability is approximately 1000, this time we reach a value of only $S = 25$. The graphs show that GASGA has higher fitness than any other GA (other than KGA). GASGA is also less wasteful than any other GA (other than the unreliable KGA): though it finds high-fitness solutions using 1000 samples, it uses only 39 samples per chromosome on average. This is exactly what we aimed for: a GA that achieves high fitness and reliability but low wastefulness.

As noted above, in further experiments using different problem parameters we obtained the same relationships among the GAs. The only difference was the SARSA(λ) result: on this instance it found a solution that was approximately as good as that found by GASGA, on others it found better solutions, and on others it found worse solutions. This illustrates the known fact that Reinforcement Learning and Evolutionary Computation are rival approaches to some problems, and neither dominates the other over all instances [16].

GASGA should find application to many problems with noisy fitness functions. The required number of samples can be chosen by considering the required solution accuracy and the observed variance in solution fitness. Parameter S must currently be tuned by hand: too small a value causes GASGA to behave like KGA, and it never obtains a reliable solution; too large a value causes it to behave like NGA, and it converges slowly. We tried automating S by maintaining it at a level that only just generates a chromosome with 1000 samples, but this forced it to a higher value than necessary (over 100); automation of S is a topic for future work.

5 Conclusion

We designed a simple new resampling strategy for steady-state GAs that makes no assumptions about fitness noise distributions (though problems with different

distributions will probably require the parameter values to be tuned differently), does not require a large population, provides a high level of reliability, yet takes a low number of samples on average. Incorporated into GENITOR and applied to a problem from classical Inventory Control, it gave better results than four other sampling strategies. In future work we will evaluate GASGA on other problems with noisy fitness functions such as perception [7], image registration [9, 15], network design [27] and remediation design [11].

None of the algorithms we tested are able to find optimal policies for the inventory problem so it is a challenging benchmark for Evolutionary Computation, and in further experiments we also found it to be hard for Reinforcement Learning and Mixed Integer Programming. This makes it an interesting benchmark despite its simplicity, and in future work we will add features such as order capacity constraints.

References

1. A. Aizawa, B. Wah. Scheduling of Genetic Algorithms in a Noisy Environment. *Evol. Comput.* 2(2):97–122, 1994.
2. D. V. Arnold, H.-G. Beyer. Local Performance of the (1+1)-ES in a Noisy Environment. *IEEE Trans. Evolutionary Computation* 6(1):30–41, 2002.
3. T. Beielstein, S. Markon. Threshold Selection, Hypothesis Tests, and DOE Methods. *Congress on Evolutionary Computation*, IEEE Press, 2002, pp. 777–782.
4. H.-G. Beyer. Evolutionary Algorithms in Noisy Environments: Theoretical Issues and Guidelines for Practice. *Computer Methods in Applied Mechanics and Engineering* 186(2–4):239–267, 2000.
5. J. Branke, C. Schmidt, H. Schmeck. Efficient Fitness Estimation in Noisy Environments. *Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, 2001, pp. 243–250.
6. L. T. Bui, H. A. Abbass, D. Essam. Fitness Inheritance for Noisy Evolutionary Multi-Objective Optimization. *Genetic and Evolutionary Computation Conference*, Washington DC, USA, ACM Press, 2005.
7. G. de Croon, M. F. van Dartel, E. O. Postma. Evolutionary Learning Outperforms Reinforcement Learning on Non-Markovian Tasks. *Workshop on Memory and Learning Mechanisms in Autonomous Robots, 8th European Conference on Artificial Life*, Canterbury, Kent, UK, 2005.
8. P. J. Darwen. Computationally Intensive and Noisy Tasks: Coevolutionary Learning and Temporal Difference Learning on Backgammon. *Congress on Evolutionary Computation*, 2000.
9. J. M. Fitzpatrick, J. J. Grefenstette. Genetic Algorithms in Noisy Environments. *Machine Learning* 3:101–120, 1988.
10. D. E. Goldberg, K. Deb, J. H. Clark. Genetic Algorithms, Noise, and the Sizing of Populations. *Complex Systems* 6:333–362, 1992.
11. G. Gopalakrishnan, B. S. Minsker, D. Goldberg. Optimal Sampling in a Noisy Genetic Algorithm for Risk-Based Remediation Design. *World Water and Environmental Resources Congress*, ASCE, 2001.
12. E. J. Hughes. Evolutionary Multi-objective Ranking with Uncertainty and Noise. *First International Conference on Evolutionary Multi-Criterion Optimization, Lecture Notes In Computer Science* vol. 1993, Springer-Verlag, 2001, pp. 329–343.

13. Y. Jin, J. Branke. Evolutionary Optimization in Uncertain Environments — a Survey. *IEEE Transactions on Evolutionary Computation* 9(3):303–317, 2005.
14. B. L. Miller. Noise, Sampling, and Efficient Genetic Algorithms. PhD thesis, University of Illinois, Urbana-Champaign, 1997.
15. B. L. Miller, D. E. Goldberg. Optimal Sampling for Genetic Algorithms. *Intelligent Engineering Systems Through Artificial Neural Networks*, vol. 6, ASME Press, 1996, pp. 291–298.
16. D. E. Moriarty, A. C. Schultz, J. J. Grefenstette. Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research* 11:241–276, 1999.
17. A. Di Pietro, L. While, L. Barone. Applying Evolutionary Algorithms to Problems With Noisy, Time-Consuming Fitness Functions. *Congress on Evolutionary Computation*, IEEE, 2004, pp. 1254–1261.
18. Y. Sano,H. Kita. Optimization of Noisy Fitness Functions by Means of Genetic Algorithms Using History of Search With Test of Estimation. *Congress on Evolutionary Computation*, IEEE, 2002, pp. 360–365.
19. C. Schmidt, J. Branke, S. E. Chick. Integrating Techniques from Statistical Ranking into Evolutionary Algorithms. *EvoWorkshops, 3rd European Workshop on Evolutionary Algorithms in Stochastic and Dynamic Environments, Lecture Notes in Computer Science* vol. 3907, Springer, 2006, pp. 752–763.
20. E. A. Silver, D. F. Pyke, R. Peterson. Inventory Management and Production Planning and Scheduling. John-Wiley and Sons, New York, 1998.
21. J. B. Smalley, B. Minsker, D. E. Goldberg. Risk-Based In Situ Bioremediation Design Using a Noisy Genetic Algorithm. *Water Resour. Res.* 36(10):3043–3052, 2000.
22. P. Stagge. Averaging Efficiently in the Presence of Noise. *5th International Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science* vol. 1498, Springer-Verlag, 1998, pp. 188–197.
23. P. D. Stroud. Kalman-Extended Genetic Algorithm for Search in Nonstationary Environments with Noisy Fitness Functions. *IEEE Transactions on Evolutionary Computation* 5(1):66–77, 2001.
24. R. S. Sutton, A. G. Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.
25. T. W. Then, E. K. P. Chong. Genetic Algorithms in Noisy Environments. *9th IEEE International Symposium on Intelligent Control*, Columbus, Ohio, USA, 1994, pp. 225–230.
26. D. Whitley, J. Kauth. GENITOR: A Different Genetic Algorithm. *Rocky Mountain Conference on Artificial Intelligence*, Denver, CO, USA, 1988, pp. 118–130.
27. J. Wu, C. Zheng, C. C. Chien, L. Zheng. A Comparative Study of Monte Carlo Simple Genetic Algorithm and Noisy Genetic Algorithm for Cost-Effective Sampling Network Design Under Uncertainty. *Advances in Water Resources* 29:899–911, 2006.