A Comparison of Sampling Strategies for Parameter Estimation of a Robot Simulator

Gordon Klaus, Kyrre Glette, and Jim Tørresen

University of Oslo, Norway
Department of Informatics
{gordonk,kyrrehg,jimtoer}@ifi.uio.no

Abstract. Methods for dealing with the problem of the "reality gap" in evolutionary robotics are described. The focus is on simulator tuning, in which simulator parameters are adjusted in order to more accurately model reality. We investigate sample selection, which is the method of choosing the robot controllers, evaluated in reality, that guide simulator tuning. Six strategies for sample selection are compared on a robot locomotion task. It is found that strategies that select samples that show high fitness in simulation greatly outperform those that do not. One such strategy, which selects the sample that is the expected fittest as well as the most informative (in the sense of producing the most disagreement between potential simulators), results in the creation of a nearly optimal simulator in the first iteration of the simulator tuning algorithm.

Keywords: the reality gap, evolutionary robotics, simulation.

1 Introduction

Evolutionary robotics (ER) [1,2], the application of evolutionary algorithms (EAs) to robot design, has shown itself to be a powerful technique. The ability of EAs to find novel solutions in a large or unfamiliar search space has been demonstrated, e.g., with Sims' swimming robots [3] and in antenna design [4]. Using ER, the designer is free to explore otherwise daunting domains like the complex dynamics of tensegrity structures and soft materials [5,6,7] or the space of robot morphologies [7,8,9]. A simple demonstration of the advantage of ER over the hand-design of robots is given in [10].

While it is possible to do evolutionary robotics in the real world [11], this can be a very time consuming task. The evaluation of a single candidate solution can take on the order of tens of seconds, not including setup time, and an entire run of an EA typically involves thousands of such evaluations. The benefit of using a simulator, which, given sufficient computing power, can perform many evaluations in the time it would take to perform one in reality, is palpable.

This speedup, however, comes with a cost. Because a simulator is only an approximation of reality, it necessarily changes the problem being solved. An individual that behaves a certain way in simulation may not behave the same when transferred to reality. As a result, the fitness landscape will be different.

Most importantly, optima of the approximated fitness function may not be optimal in reality. To deal with this problem of the "reality gap" (as it has been called) several approaches have been made.

It was shown that, by simulating only those aspects of reality that are relevant to the problem at hand (i.e., building "minimal" simulations) and by using empirically determined amounts of noise to model noisy or poorly understood aspects of a system, the transferability to reality of robot controllers evolved in simulation can be improved [12,13,14]. For example, sensors and actuators do not behave in an idealized fashion; there is always some noise, and it can have a large effect on the functioning of a robot. Moreover, the nondeterminism of a noisy simulator helps to produce robust controllers that are less likely to take advantage of a simulator's idiosyncrasies; i.e., it discourages "cheating".

Rather than concerning oneself greatly with the quality of the simulator, it is alternatively possible to sidestep the problem by simply accepting that a simulator has inaccuracies. The transferability approach [15,16] uses multi-objective optimization to explicitly consider the trade-off between fitness and transferability. Other methods involve designing adaptive controllers that can cope with differences between simulation and reality. In [17], a mobile robot was able to perform a non-trivial task in reality after being very rapidly evolved in a simulator, using Hebbian rules to develop a neural controller on the fly. In [18], a control architecture was developed which enabled an evolved robot to adaptively anticipate errors between its expected and actual motions, allowing it to recover from perturbations not encountered in simulation.

Each of these methods is promising in its own way, but they are not the focus of this paper. Here, we deal with methods of improving the quality of simulation. A simulator is typically configured by some set of parameters. For example, a physics simulator has parameters related to friction and restitution that govern the interactions of different materials, and parameters for the dimensions and mass distributions of objects being simulated. Many values can be obtained by direct measurement. However, given that a simulator is a simplification of the real world, there may not be direct correspondences between real and simulated parameters – for example, when nonlinear mechanisms like robot actuators are modeled linearly or when detailed objects are represented by coarse shapes, as is often done.

It is thus necessary to adjust simulation parameters such that the simulator more accurately reflects the real world. Some improvement can be made by simply hand-tuning [19] but clearly more sophisticated methods are required. Even when suitable values can be determined by measurement, a method of automatic discovery or inference could be useful, for example in an autonomous adaptive robot that maintains internally a model of its environment.

The approaches taken in Back-to-Reality (BTR) [20,21], estimation-exploration (EE) [22,23], and using sequential surrogate optimization (SSO) [24] are largely similar to each other: Two coupled optimization algorithms are run in an interleaved fashion, one to search for solutions to a primary task such as simulated robot locomotion and another to improve the accuracy of the simulator. The product of each run of the primary search (a controller for a simulated robot) is evaluated in reality and then used in subsequent simulator optimizations; and the product of each simulator optimization is used in the following primary search. (A more detailed description is given later.) Ultimately, this should result in the convergence toward accurate simulators and, as a result, robot controllers that are fit in reality.

BTR actually interleaves *three* algorithms. In addition to the two just mentioned, it includes a learning-in-reality step after each simulator search, seeded by the prior primary search, and the results of which are used to seed the next primary search. The value of this extra step is a bit unclear. It is expensive to do such fine tuning in the real world, and any information gained is then lost upon return to simulation. Even if the real-world evaluations were used to inform the simulator search, it would be a large price to pay for some tightly clustered, and thus information poor, data points.

An interesting aspect of EE is its measure of simulator fitness. While the other two methods search for a simulator that minimizes the difference between the fitnesses of some individuals in simulation and reality, EE tries to minimize the difference between the robots' sensor readings in simulation and reality. It is a more complicated calculation to perform but it also provides a great deal more information.

Perhaps the most important difference between these methods is in their specific strategies for selecting individuals for evaluation in reality. Individuals evaluated in reality provide the data points, or *samples*, used for tuning the simulator. Because the goal is to do as few real-world evaluations as necessary, it is important that sample selection be done wisely.

BTR and EE operate in the same manner, selecting for evaluation in reality the individual that is fittest in the current simulator. EE is actually described to work differently, but is only used that way in one of its four applications, and not in evolutionary robotics; This other manner is to select the individual that maximizes the disagreement between a number of candidate simulators, i.e., the most informative individual. SSO uses a hybrid strategy, selecting the fittest individual, or, if that individual is within a threshold distance of a previously selected individual, the most informative individual (based on a fitness error estimate that is maintained as part of the surrogate fitness function).

These different selection strategies seem promising but it isn't clear which would give the best results. The goal of this paper, then, is to compare them (and others) on a common task. In the next section, we describe this task, the general algorithm, and each of the specific strategies to be compared. Then, we present and discuss the results. Finally, we conclude and consider avenues for future work.

2 Implementation

The benchmark task in this investigation is a common one: to design a controller for robot locomotion. Figure 1 depicts the robot, a quadruped with a total 12 degrees of freedom in its limbs. To produce motion, each of its actuators is periodically extended and contracted based on a simple pulse shaped function determined by two parameters; a total of 24 parameters thus define such a controller. The task is then to use an EA to search for controller parameters that cause the robot to walk at the highest speed. Details of the robot, its controller, and the EA are given in [10].

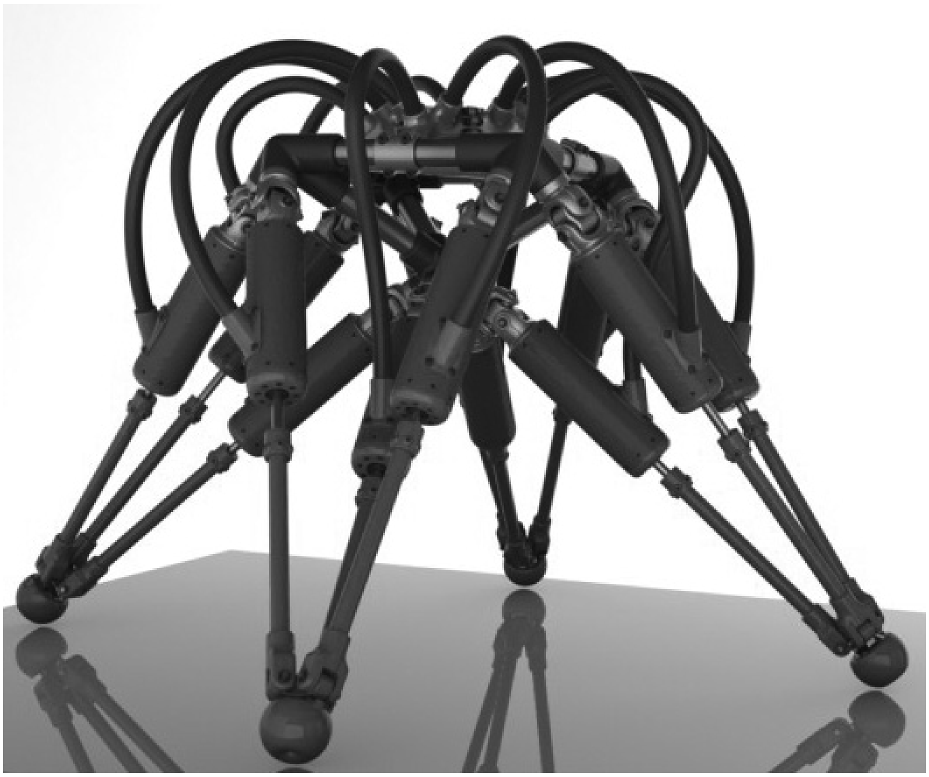

Fig. 1. A rendering of the robot used in this investigation

The ultimate goal is to be able to reliably perform this task for a robot situated in the real world, taking advantage of a simulator to perform most of the expensive robot evaluations required by the EA. In order to determine the expected relative performance of a number of different methods for achieving this goal, however, experiments need to be repeated many times due to the stochastic nature of EAs. This would require thousands of real-world robot evaluations. To expedite this investigation, we have substituted a simulation for the real world. That is, instead of doing evaluations in the real world, we do them in a simulator whose configuration is *unknown* (i.e., hidden from the algorithm). As long as the configuration of "reality" (simulated or otherwise) remains hidden from the process that generates approximations of it, we can consider this to be a "reality gap" problem. It should therefore be sufficient for the purpose of comparing selection strategies.

The PhysX software library was used to simulate robot movement, as in [10]. The simulated reality (henceforth just "reality") was configured such that the material constituting the robot and the ground plane had coefficients of static and dynamic friction 1.0 and 0.8, respectively, and restitution (bounciness) 0.0. The simulator (i.e., the approximating simulator, not "reality") was parameterized by these material properties, so that the process of simulator tuning was a search for values of these three coefficients.

The basic algorithm, illustrated in Figure 2, is essentially the same as in BTR, EE, and SSO. We iterate the following procedure: First, select a new sample (defined in the next paragraph) using one of the selection strategies described later; then, use all the samples collected thus far to optimize a new simulator. Finally, after performing some number of these iterations, the latest simulator is used to evolve a robot controller which, transferred to reality, is the final product of the algorithm. The algorithm is iterative because selection strategies typically use the tuned simulator for the subsequent sample selection; initial sample selection uses a random simulator.

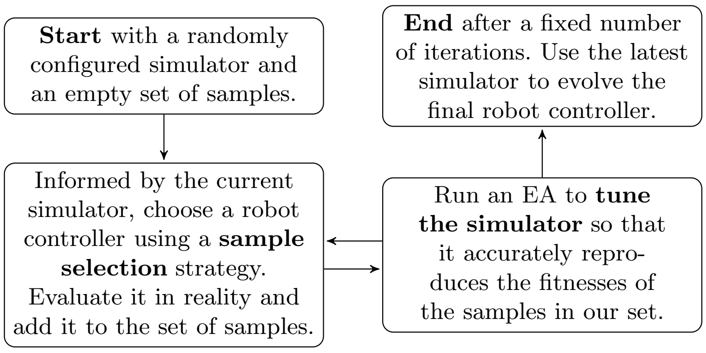
Start with a randomly configured simulator and an empty set of samples.
End after a fixed number of iterations. Use the latest simulator to evolve the final robot controller.
Informed by the current simulator, choose a robot controller using a sample selection strategy. Evaluate it in reality and add it to the set of samples.
Run an EA to tune the simulator so that it accurately reproduces the fitnesses of the samples in our set.

Fig. 2. The flow of the basic simulator tuning algorithm

A sample, as mentioned earlier, is a robot controller whose fitness (here, walking speed) we evaluate in reality. The real-world fitness values of the samples drive simulator tuning: We use an EA to find the simulator (specifically, the three parameters named earlier) that accurately reproduces the fitness values of the samples collected thus far. The fitness of a simulator *sim*, to be maximized, is

$$f(sim) = \frac{1}{\sum_{x \in samples}(f_{sim}(x) - f_{real}(x))^2}$$

where $f_{sim}(x)$ and $f_{real}(x)$ are the fitnesses in the simulator and in reality, respectively, of an individual x. We use the same EA parameters as in the robot optimization, but only 2000 evaluations.

Figure 3 illustrates the progression of the algorithm when it is run on a much simpler function optimization task. In this example, we can visualize the fitness landscape, sample selection, the successively more accurate simulators, and transferral from simulation to reality.

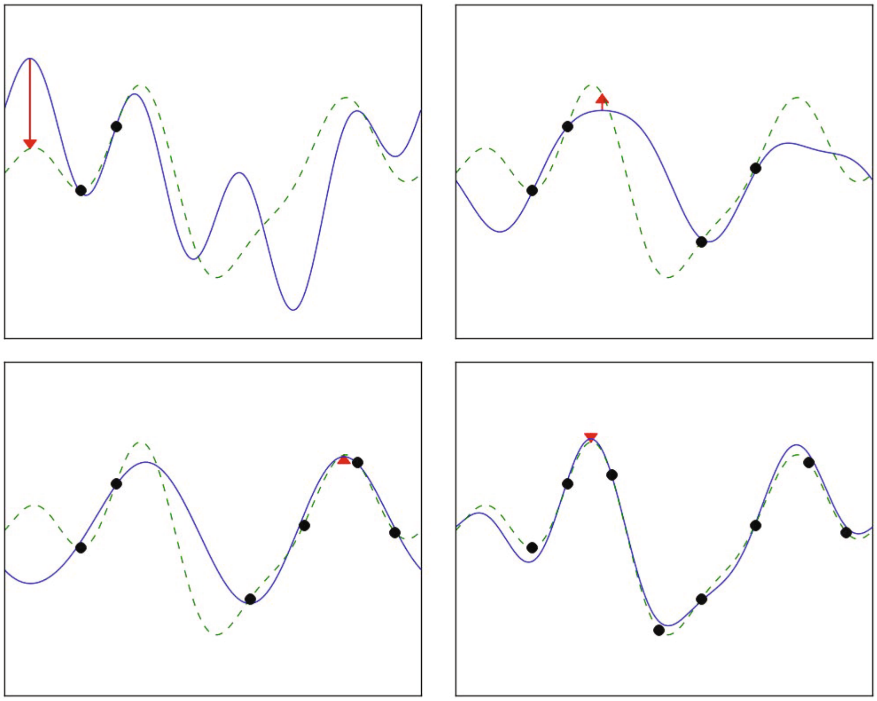

Fig. 3. A visualization of the algorithm being run on a function optimization task after 2, 4, 6, and 8 iterations. The dashed curve is the true function (reality) and the solid curves are approximating functions (simulations). Dots indicate samples. The arrow indicates the fittest in simulation and its transferral to reality. Notice that, as more samples are added, the simulations become more accurate and the fitness of the transferred individual increases until it is near the global maximum of the real function.

In the following paragraphs we describe the six sampling strategies that are the objects of comparison in this paper. The first four strategies fit in the iterative form of the algorithm described above and use the ideas from BTR, EE, and SSO, while the last two are non-iterative and much simpler.

Simulated Fit. This is the strategy used in BTR and EE and partially in SSO. The individual is selected for evaluation in reality that has the highest fitness in the latest tuned simulator (or, in the first iteration, a simulator with randomly chosen parameters). An EA is run to find this individual; this EA uses the same parameters as the one used to produce the final result of the algorithm. This strategy is motivated by the idea that our simulator needs to accurately model only individuals of high fitness; it thus tries to bias the samples towards what are estimated to be fit individuals.

Simulated Fit and Unique. This is the same as the previous strategy, but with a modification to the fitness function to maintain some distance between the samples. The modified fitness function is

$$f^*_{sim}(x) = f_{sim}(x) - \frac{1}{100N} \sum_{y \in samples} \frac{1}{d(x,y)}$$

where $f_{sim}(x)$ is the simulated fitness of x, $N = |samples|$, and $d(x, y)$ is the Euclidean distance between individuals x and y in terms of both genome and fitness. This strategy is motivated by the fact that we might expect the previous strategy to pick, after several iterations, very similar individuals near a local optimum. Several such tightly clustered samples would provide little more information than a single one.

Informative. This strategy was suggested in EE and used partially in SSO. The individual is selected that maximizes the disagreement between a number of simulators. A modification to the basic algorithm is required: Instead of evolving a single simulator, a diverse population of 20 simulators is evolved. To maintain diversity, the EA's method of replacement is changed: After a newly evaluated simulator is added to the population, instead of dropping the least fit, the pair of simulators whose genomes are most similar is found and the less fit of the two is removed from the population.

To find the individual that maximizes the disagreement between the simulators, an EA is used. Its fitness function is calculated as the weighted standard deviation of the fitness values the simulators produce for a given individual. The weights are the fitnesses of the simulators themselves, so that an inaccurate simulator contributes less to this measure than an accurate simulator.

Simulated Fit and Informative. This is a mixture of two strategies. It works the same as the *Informative* strategy, but in addition to the weighted standard deviation, its fitness function includes the weighted average of the fitness values produced by the diverse population of simulators.

Random. Individuals are picked with genomes drawn from a uniform distribution. As each selection is entirely independent of the previous ones, the algorithm collapses to a non-iterative form where all the samples are selected at once before a single simulator is finally tuned.

Known Fit. This strategy is like *Random* except the samples are generated as mutated versions of an individual known to have middling fitness. In this case, the hand-designed controller from [10] was used; it achieved a fitness of 0.73 in "reality". Values drawn randomly from a uniform distribution on $[0, 0.25)$ were added to each element of this controller's genome to produce each new sample. This strategy is motivated by the observation that many of the purely randomly generated samples had very low fitness. In SSO, a similar strategy was used to generate a small set of initial samples.

3 Experiments and Results

The aim is to achieve the highest real-world fitness while doing the fewest real-world evaluations. It is on this basis that we compare the performance of the different selection strategies. The creation of a generally accurate simulator may be an intermediate goal; however, a relatively poor simulator is perfectly acceptable if it enables us to find robot controllers that are very fit in reality – which may very well be the case for a simulator that captures only certain essential aspects of reality. For this reason, we make no explicit judgments of simulator accuracy.

For each of the strategies, the algorithm is iterated ten times; that is, ten real-world evaluations are performed. After each iteration of the algorithm, the best simulator is used to evolve a robot controller for walking speed. This controller is then evaluated in reality and its fitness (or the maximum fitness of the samples, if it is larger) is recorded. We do not count this real-world evaluation among those that are the basis for comparison, as it is not used as a sample for simulator tuning – it is only a view into the algorithm's performance. Figure 4 shows the recorded fitness values averaged over 40 repetitions of this process.

In addition to comparing the different strategies to each other, we should also consider them with regard to the expected performance of evolving directly in reality. Because we have substituted the real world with a simulator, it is a simple task to compute the fitness expected from evolving directly in reality. Just as in [10], robot controllers were evolved for 10000 evaluations in the simulator configured as "reality"; with 100 repetitions, the best fitness at the end of the evolutionary runs had an average of 1.33 (standard deviation 0.10).

All of the strategies showed real-world fitness increasing with the number of real-world evaluations. Before any iterations, all strategies of course produced roughly the same expected fitness, about 0.9, from evolving in a randomly configured simulator. With more iterations, fitnesses improved (diminishingly) to more or less 1.3.

Known fit performed only marginally better than *Random*. More improvement could probably have been achieved by using a fitter individual as the seed for generating the samples.

The three strategies that involve simulated fitness were significantly better than the others in terms of both average and standard deviation of real-world fitness. After only three real-world evaluations, these three strategies achieved real-world fitness of at least 1.3, and after ten iterations they reached 1.4. Standard deviations were in the range 0.10-0.15, about three times smaller than in the other strategies and comparable to that found when evolving directly in reality. This indicates that the inclusion of simulated fitness as a factor in sample selection is very effective, perhaps essential.

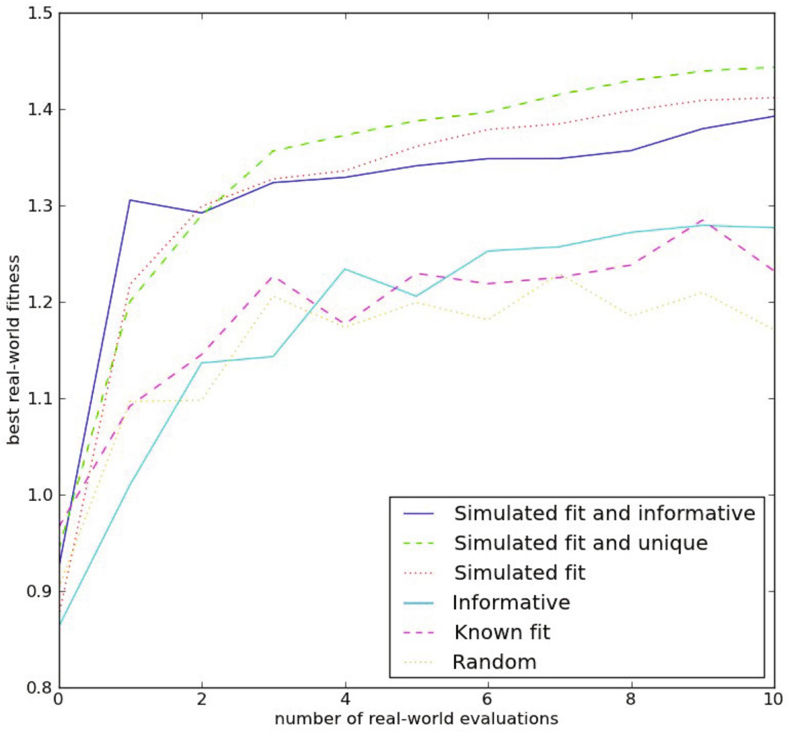

Fig. 4. Fitness in reality versus number of real-world evaluations for the six sample selection strategies. The statistical significance of the results (using Student's t-test) is as follows: With just one real-world evaluation, the *Simulated fit and informative* strategy was significantly different from the other strategies ($p < 0.01$). Combining the data for more than two real-world evaluations, each of the strategies involving simulated fitness was significantly different from the others ($p << 0.001$) and *Known fit* was significantly different from *Random* ($p \approx 0.017$). Considering more than five real-world evaluations, each of the three weakest strategies was significantly different from the others ($p < 0.05$).

The additional criterion of uniqueness appears to give a small improvement over plain *Simulated fit* after a couple of iterations, as expected. We might expect the same sort of improvement from *Informative*, as informative implies uniqueness (a repeated sample is totally uninformative), but we do not see it. This is because the strategies involving simulated fitness produced nearly perfectly tuned simulators after about 3 or 4 iterations, as they have attained the same fitness as evolution in reality using the same number of simulated evaluations. Beyond these 3 or 4 iterations, the algorithm is hardly tuning the simulator anymore, instead just searching for fitter samples; the increasing fitness is similar to what would be seen by simply extending the EAs beyond 10000 evaluations.

That the different strategies show different rates of improvement in later iterations is due to their specific construction: The strategy that promotes unique samples introduces a weak exploratory force in the algorithm, leading it to search locally and thus to improve more quickly, while the strategy biased towards informative samples introduces a stronger exploratory force (informative samples will typically be quite distant from each other) that may lead the algorithm away from regions of high fitness.

Most striking is the performance of the *Simulated fit and informative* strategy. While *Informative* alone was only slightly better than *Random* and *Known fit*, in combination with *Simulated fit* it produced a nearly perfect result in just a single iteration. The fact that the other two strategies outperform it in later iterations is due to the phenomenon described it the previous paragraph, and for this reason shouldn't be considered a weakness of this strategy. This is the most promising of the six selection strategies.

It is interesting to consider how poorly the *Informative* strategy performed, especially when its combination with *Simulated fit* was so successful. It seems likely that the diversity maintenance mechanism used in simulator evolution was not as effective as desired – it plays no role in the first iteration (in which randomly configured simulators are used), where *Simulated fit and informative* really shines.

4 Conclusion and Future Work

Our investigation has demonstrated the important role that sample selection strategies play in simulator tuning. Of the six strategies used here, the *Simulated fit and informative* strategy proved to be the most successful at closing the "reality gap" for this particular task of robot locomotion. In fact, seeing as the algorithm was able to nearly perfectly tune the simulator in just a single iteration using this strategy, it seems reasonable to say that this strategy has *solved* this particular instance of the task.

It remains to be seen how well this algorithm scales up to more complex scenarios; this will be the main focus of our future work. This instance of simulator tuning was rather simple, involving the estimation of only three parameters. Other parameters to be considered include motor forces, noise sources, and dimensions and mass distribution of the robot body. Ultimately, testing must be done on a physical robot outside simulation.

To succeed at more complex tasks, it may be necessary to improve upon the *Simulated fit and informative* strategy. The current implementation involves a simple sum of two factors (one from simulated fitness, the other informative) whereas this probably ought to be a more general weighted sum so that it is possible to adjust the influence of the two factors. It may also be beneficial to more tightly interleave the evolution of diverse simulators with the evolution of informative samples, as their respective diversity and informativeness are strongly coupled.

Finally, as sample selection strategies become more sophisticated, it could be practical to consider a more nuanced basis for comparison than just the number of real-world evaluations. If some strategies take much longer to run than others then the total running time would be a better measure. If a strategy takes a very long time to run then one might consider instead to do more real-world evaluations in that time; to capture this possibility, the time it takes to perform real-world evaluations must be considered. In this situation it would probably make sense to give greater weight to time spent doing real-world evaluations than time spent computing, as it demands more constant human attention.

References

1. Doncieux, S., Bredeche, N., Mouret, J.-B.: Exploring new horizons in evolutionary design of robots. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE Press (2009)
2. Harvey, I., Husbands, P., Cliff, D., Thompson, A., Jakobi, N.: Evolutionary robotics: the sussex approach. Robotics and Autonomous Systems 20, 205–224 (1997)
3. Sims, K.: Evolving virtual creatures. In: SIGGRAPH 1994: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, pp. 15–22. ACM, New York (1994)
4. Linden, D., Hornby, G., Lohn, J., Globus, A., Krishunkumor, K.: Automated antenna design with evolutionary algorithms. American Institute of Aeronautics and Astronautics 5, 1–8 (2006)
5. Rieffel, J., Trimmer, B., Lipson, H.: Mechanism as mind: What tensegrities and caterpillars can teach us about soft robotics. In: Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems (2008)
6. Glette, K., Hovin, M.: Evolution of Artificial Muscle-Based Robotic Locomotion in PhysX. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2010)
7. Rieffel, J., Saunders, F., Nadimpalli, S., Zhou, H., Hassoun, S., Rife, J., Trimmer, B.: Evolving soft robotic locomotion in PhysX. In: GECCO 2009: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference, pp. 2499–2504. ACM, New York (2009)
8. Bongard, J.C.: Incremental Approaches to the Combined Evolution of a Robot's Body and Brain. PhD thesis, University of Zurich (2003)
9. Macinnes, I., Di Paolo, E.: Crawling out of the simulation: Evolving real robot morphologies using cheap reusable modules. In: Pollack, J., Bedau, M., Husbands, P., Ikegami, T., Watson, R. (eds.) Artificial Life IX: Proceedings of the Ninth Interational Conference on the Simulation and Synthesis of Life, pp. 94–99. MIT Press, Cambridge (2004)
10. Klaus, G., Glette, K., Høvin, M.: Evolving Locomotion for a Simulated 12-DOF Quadruped Robot. In: Lones, M.A., Smith, S.L., Teichmann, S., Naef, F., Walker, J.A., Trefzer, M.A. (eds.) IPCAT 2012. LNCS, vol. 7223, pp. 90–98. Springer, Heidelberg (2012)
11. Zykov, V., Bongard, J.C., Lipson, H.: Evolving dynamic gaits on a physical robot. In: Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO (2004)
12. Jakobi, N.: Minimal Simulations for Evolutionary Robotics. PhD thesis, University of Sussex (1998)

13. Jakobi, N., Husbands, P., Harvey, I.: Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics. In: Morán, F., Merelo, J.J., Moreno, A., Chacon, P. (eds.) ECAL 1995. LNCS, vol. 929, pp. 704–720. Springer, Heidelberg (1995)
14. Miglino, O., Lund, H.H., Nolfi, S.: Evolving mobile robots in simulated and real environments. Artificial Life 2, 417–434 (1996)
15. Koos, S., Mouret, J.-B., Doncieux, S.: The transferability approach: Crossing the reality gap in evolutionary robotics. IEEE Transactions on Evolutionary Computation (2012)
16. Koos: The transferability approach- an answer to the problems of reality gap, generalization, and adaptation. PhD thesis, Institut des Systémes Intelligents et de Robotique Université Pierre et Marie CURIE (2011)
17. Floreano, D., Urzelai, J.: Evolution of Plastic Control Networks. Autonomous Robots 11(3), 311–317 (2001)
18. Hartland, C., Bredeche, N.: Evolutionary robotics, anticipation and the reality gap. In: IEEE International Conference on Robotics and Biomimetics, ROBIO 2006, pp. 1640–1645 (December 2006)
19. Glette, K., Klaus, G., Zagal, J.C., Tørresen, J.: Evolution of locomotion in a simulated quadruped robot and transferral to reality. In: Artificial Life and Robotics (2012)
20. Zagal, J.C., Ruiz-del-Solar, J., Vallejos, P.: Back to reality: Crossing the reality gap in evolutionary robotics. In: Proceedings of IAV 2004, the 5th IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal (2004)
21. Zagal, J.C., Ruiz-Del-Solar, J.: Combining simulation and reality in evolutionary robotics. J. Intell. Robotics Syst. 50, 19–39 (2007)
22. Bongard, J.C., Lipson, H.: Once more unto the breach: co-evolving a robot and its simulator. In: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9), pp. 57–62 (2004)
23. Bongard, J., Lipson, H.: Nonlinear system identification using coevolution of models and tests. IEEE Transactions on Evolutionary Computation 9, 361–384 (2005)
24. Hemker, T., Sakamoto, H., Stelzer, M., Stryk, O.V.: Hardware-in-the-loop optimization of the walking speed of a humanoid robot. In: CLAWAR 2006: 9th International Conference on Climbing and Walking Robots, pp. 614–623 (2006)