Honda Research Institute EU
Honda Research Institute Europe GmbH
https://www.honda-ri.de/

The Impact of Variable Run Times on Evolutionary Optimization Algorithms

Tobias Rodemann

2013

Preprint:

This is an accepted article published in ITI Symposium. The final authenticated version is available online at: https://doi.org/[DOI not available]

The Impact of Variable Run-Times on Evolutionary Optimization Algorithms

Tobias Rodemann
Honda Research Institute Europe GmbH, Offenbach/Main

Kurzfassung

Evolutionäre Algorithmen sind erprobte Ansätze fuer die Optimierung komplexer, technischer Systeme. Ein wesentlicher Vorteil ist die leichte Parallelisierbarkeit, wodurch eine massive Reduktion der Optimierungszeit erreicht werden kann. Dieser Vorzug geht teilweise verloren, wenn die Rechenzeit einer Systemevaluierung hohe Fluktuationen aufweist, d.h. die Optimierung auf die langsamste Rechnung warten muss. In dieser Arbeit diskutiere ich dieses Problem für verschiedene Typen von evolutionären Algorithmen und evaluiere einen bestimmten Lösungsansatz – die harte Terminierung langsam laufender Prozesse.

Abstract

Evolutionary algorithms are well established methods for the optimization of complex technical systems. One of their main benefits is easy parallelization, leading to a massive reduction in the time needed for optimization. This advantage is lost when run-times of fitness evaluations show a high variation, i.e. the optimization program has to wait for the completion of the slowest run. In this work I will discuss this problem for different types of evolutionary optimization algorithms and evaluate a specific solution: terminating long-running evaluations.

Section 1: Introduction and Motivation

Evolutionary optimization algorithms use nature-inspired approaches to find optimal configurations of complex systems which are often difficult to tune manually. These algorithms search to some degree randomly in the space of possible configurations, but preferentially look for configurations close to good ones already found. The specific realization is a process where successful (high quality) solutions are taken as parents and offspring are generated by mutation and recombination of those parents. This simple approach makes Evolutionary Algorithms (EAs) very attractive since very little domain knowledge is required. Another substantial benefit

is that the evaluation of candidate solutions can be parallelized. Often, a fully linear speed-up is possible (up to the number of offspring).
This enormous speed-up potential rests on the assumption that many evaluations started in parallel will finish (roughly) at the same time. However, this is not true in general for tools like SimulationX, where run times may vary substantially. We encountered this problem in the optimization of a smart home system [4], where we did not observe a speed-up in optimization time that came even close to the parallelization factor. We observed a very high variability in the run-times of the simulation. An example is shown in Fig. 1 for 10000 evaluated configurations. All shown results are generated on a 32-bit Linux desktop computer using C-code export. Since the efficiency of the parallelization is mostly inhibited by a few large outliers, it is tempting to simply remove those from the optimization process, by stopping the evaluation and assigning a very bad fitness value to those solutions. Effectively, run-time is added as an additional constraint or objective. We have tested the above proposed methods and compared optimization performance and total optimization run-time. The analysis was done for an abstract optimization problem (Section 3) and the specific smart home system [4] we are working on (Section 4).

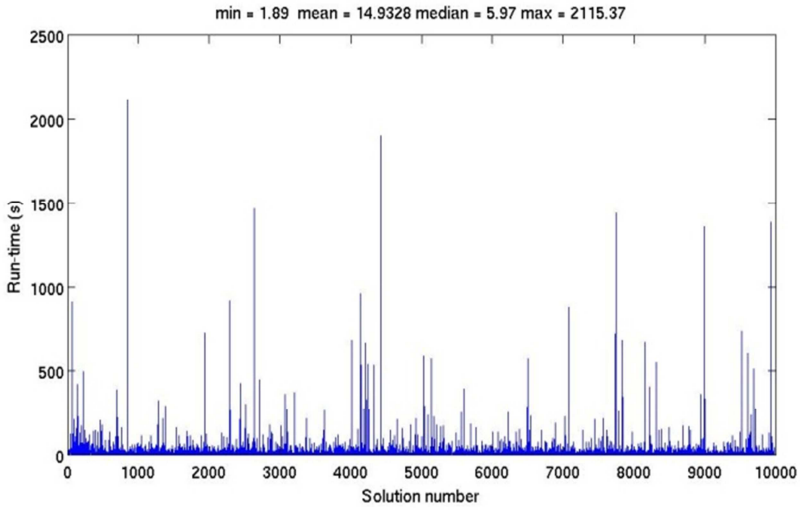


Fig. 1: Simulation run-times (10000 variations tested)

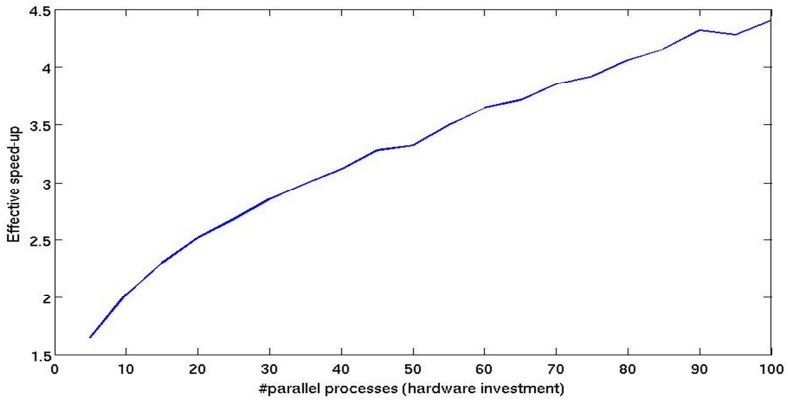


Fig. 2 Effective Speed-up Factor

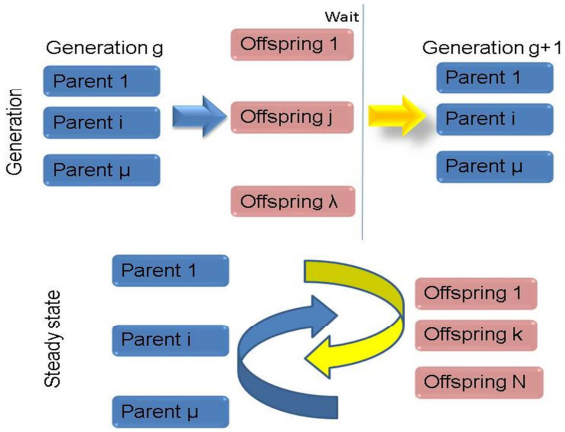


Fig. 3: Generation-based vs. Steady-State Algorithms

Section 2: Evolutionary Algorithms

Most evolutionary algorithms (EAs) use one of two different approaches for generating offspring and selecting the future parents from among this offspring: generation-based and steady-state algorithms. These two alternatives are explained in Fig. 3 and will be shortly outlined below (see also [1,2,3]). For the smart home test case, which is a many-objective optimization problem, we have used the generation-based CDAS method [7] and the steady-state method SMS-EMOA [6]. Generation-based approaches use two populations, parents and offspring. Offspring are generated from the parent population by selecting parents, and changing them through mutation and recombination. Then all offspring are evaluated. This part of the optimization (which in practical applications

consumes most of the computing resources) can be easily parallelized. The drawback is that the optimization process halts until all offspring solutions have been evaluated. Steady-state algorithms use a different approach. Whenever computing resources are available, a new offspring from the current parent population is generated and evaluated. The parent population can be updated based on information from any evaluated offspring. It is not necessary to wait for completion of all solutions. A large difference becomes visible when parallel evaluation of solutions and variable computing times are added. As an example consider the data shown in Fig. 1. The effective speed-up when running a variable number of parallel processes in a generation-based approach is shown in Fig. 2. The data shows that even when running the evaluation on 100 processes (cores) in parallel, the total run-time will only be decreased by a factor of 4.5 – hardly worth the huge extra cost. In an extreme situation, a single, long-running evaluation might nullify all benefits of parallel computing. The steady-state algorithm on the other hand will continue even if some evaluations run for long times, unless those evaluations block all available resources. Therefore steady-state methods are in general more computational efficient. However they are working on a different optimization problem. When differences in evaluation time become very large, steady-state algorithms will primarily be guided by the quick running solutions and underrepresent information provided by long-running solutions.

An obvious option for solving the resource usage problem in generation-based methods (and to some degree also in steady-state methods), is to interrupt evaluations exceeding a hard threshold and assign a very bad fitness value to those solutions, removing them from the parent population. Steady-state methods already have this advantage to some degree but would benefit because long-running evaluations could not block computer resources.

It is important to recognize that this computational efficiency gain comes at a cost. Obviously, some potential solutions will not be considered. We need to analyze the correlation between run-times plus design parameters and objectives. In the following two sections we will look at this issue for two different tasks.

Section 3: Simplified Optimization Example

In this section we look at the case of a very simple optimization problem:

$$\min F = \sum_i \left(x_i^2\right) + 1$$

for 10 dimensions. This function is very simple because the minimum is well defined (x = [0,0]), virtually any run of an Evolutionary Strategy (ES) will find it and optimization results are therefore easy to compare. We employ the most basic version of a (μ,λ)-ES (see Rechenberg [5]), but assume that computing time T is derived as minimum run time T0 (= 1 s) plus:

1. Purely random variation **(independent).**
2. Runtime proportional to fitness. Consider that lower fitness values are better. Therefore good solutions run faster **(propFitness).**
3. Runtime inversely proportional to fitness – now good solutions are taking more computation time **(propInvFitness).**

The ES uses μ = 3 parents and λ = 12 offspring for 100 generations. Initial solutions were randomly generated. We simulated a parallel execution of all λ = 12 offspring evaluations and stopped processing at ΔT seconds (all unfinished runs got a fitness of 1000). All shown results are averages over 20 independent runs.

One has to be careful with generalizing the results, as they depend on the specific fitness function, the amount of noise, and other parameters like T0. Nevertheless, the shown results (Fig. 4) give a few examples of possible effects.

In all cases except for **independent** increasing ΔT improves the quality of the output (lower fitness values), while **independent** seems to be unaffected by the choice of ΔT (beyond very low values). The final solution quality after 100 generations is similar, except for **propInvFitness** where the final fitness is substantially higher. We also observe that total system run-time increases - only weakly for **propFitness** and strongly for **propInvFitness** with **independent** in between. We see that imposing a limit ΔT on run-times can have varying effects depending on the relation between parameters, fitness and run-time.

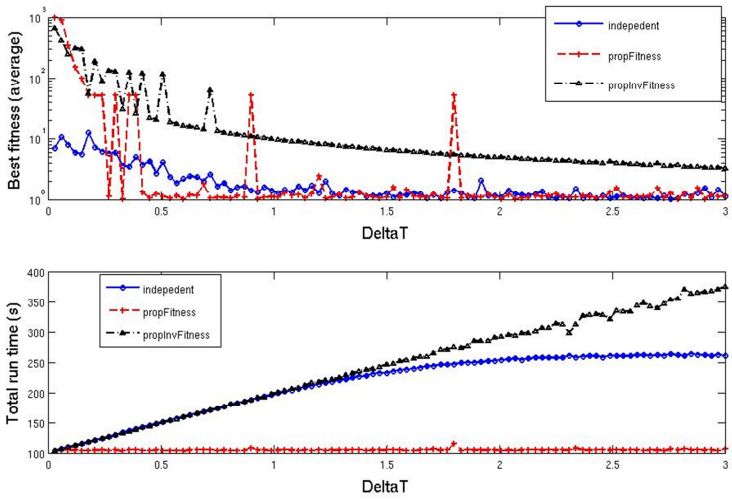


Fig. 4: Performance vs. ΔT (above T0) for different run-time thresholds. For ΔT = 3 s we got the following best fitness: **independent** = 1.151; **propFitness** = 1.06, **propInvFitness** = 3.3535.

Section 4: Results for Smart Home Optimization

Last year we presented a study for a many-objective optimization of a smart home system configuration [4] using the Green Building library [8]. In total 10 parameters were optimized considering 5 objectives. The run-times shown in Fig. 1 are measured for an older version of the system model and Green Building library. Maximum run-times were very large and the chance of exceeding the ΔT limit strongly depended on the simulated day of the year. In cooperation with ITI and EA Systems Dresden we simplified and improved the model system to reduce run-times. The result was a strong reduction in average simulation times, much fewer runs exceeding the limit time ΔT, and a different run-time pattern for the different days.

In this article we have reduced the complexity by simulating only a single day, with a run-time that is in the order of 3 s (median). We investigated the relation between total run-times and the overall system performance. Since our optimization uses 5 objectives we employ the so-called Hypervolume [6] as a performance indicator. The quality of solutions showed only a very small dependency on ΔT. This is consistent with the very low correlation coefficients between the different system parameters and objective values and run-times as can be seen from Fig. 5. Both findings suggest that, effectively, our smart home optimization seems to

have a run-time correlation best approximated by **independent**, which means that termination of long running solutions would probably have no negative impact on performance.

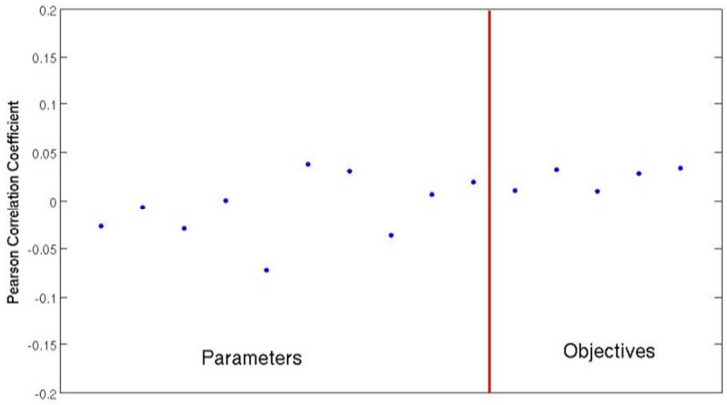


Fig. 5: Correlation coefficient between 10 input parameters (left) plus 5 objectives (right) and simulation run-time

Section 6: Summary and Conclusion

Varying computing times pose a challenge to the usage of evolutionary algorithms by a strong reduction in the effect of parallelization. An often used approach is the termination of long-running solutions, which are removed from the parent population. We have shown for a simple example that the impact of terminating the evaluation after a certain threshold depends on the way parameters and objectives are related to run-times. Both positive (reduced total-run-time) and negative (reduced solution quality) effects might be observed, where the trade-off is controlled by the time-out parameter. For our specific application case we observed a large variation in run-times with no (simple) correlation between parameters, objectives, and run-times. As a result we also found only a small reduction in optimization performance at substantially reduced total optimization time when enforcing a maximum run-time.

Bibliography:

[1] J. J. Durillo, A. J. Nebro, F. Luna, and E. Alba: *On the Effect of the Steady-State Selection Scheme in Multi-Objective Genetic Algorithms*, Proceedings of EMO 2009

[2] M. Yagoubi, L. Thobois, and M. Schoenauer: *An asynchronous steady-state NSGA-II algorithm for multi-objective optimization of Diesel combustion,* 2nd International Conference on Engineering Optimization, 2010

[3] R. Enache, B. Sendhoff, M. Olhofer, M. Hasenjäger: *Comparison of Steady-State and Generational Evolution Strategies for Parallel Architectures,* proceedings of PPSN VIII, 2004
[4] T. Rodemann and T. Schwan: *Modeling and Optimization of a Smart Home System,* ITI Symposium 2012
[5] I. Rechenberg, Evolutionsstrategie '94.Frommann Holzboog 1994
[6] N. Beume, B. Naujoks, and M. Emmerich. "SMS-EMOA: Multiobjective selection based on dominated hypervolume." European Journal of Operational Research 181.3 (2007)
[7] H. Sato, H.E. Aguirre, K. Tanaka: Controlling Dominance Area of Solutions and Its Impact on the Performance of MOEAs. Proceedings of EMO 2007
[8] http://www.itisim.com/simulationx/industries/green-building/