INVESTIGATION INTO THE BEHAVIOUR OF AN ASYNCHRONOUS POPULATION BASED HILL CLIMBING ALGORITHM

Lars Nolle and Jens Werner
Department of Engineering Science
Jade University of Applied Science
Friedrich-Paffrath-Straße 101
26389 Wilhelmshaven, Germany
Email: {lars.nolle|jens.werner}@jade-hs.de

KEYWORDS
Parallel processing, Online-monitoring, Optimisation, Hill Climbing, Asynchronicity

ABSTRACT

Previously, a variation of Asynchronous Population Based Hill Climbing was applied to a discrete optimisation task from the field of electronic circuit design with a challenging 9 dimensional search space. The algorithm exhibited a certain behaviour, which is analysed in this new research. A problem was the asynchronous nature of the search algorithm, which did not allow the search threads to save the internal state data into log files. Instead, a novel monitoring strategy had to be developed, which accesses this internal data via shared memory in specified times intervals. It was possible to show that the algorithm under investigation is capable of switching between exploration and exploitation of the search space during the search. This prevents the algorithm from being stuck in a local optimum.

INTRODUCTION

In modern engineering it is usually important to design systems that not only satisfy the functional requirements but are also as cheap as possible. Here, computational optimisation methods can help to find optimal designs automatically (Nolle et al. 2016). Various direct search methods have been proposed in the literature, for example Genetic Algorithms (GA) (Holland 1975, Goldberg 1989), Simulated Annealing (SA) (Kirkpatrick et al. 1984), Particle Swarm Optimisation (PSO) (Kennedy and Eberhart 1995), or Ant Colony Optimisation (ACO) (Dorigo and Gambardella 1997). Such algorithms start with initial solutions, which are stepwise refined according to a method-dependent strategy. A large number of these refinements are typically needed to arrive at a near-optimal solution. In this process, the bottleneck is often the evaluation of the quality, or fitness, of a new solution. A single evaluation might take anything from minutes to days (Emmerich et al. 2002). Since many modern computers have more than one computing core, parallel computing could offer a solution to this problem.

Parallel systems can be classified using Flynn's taxonomy (Flynn 1972). Modern multi-core processors can run different programs simultaneously, processing different data. Hence they fall into the Multiple-Instruction Multiple-Data (MIMD) category of Flynn's taxonomy. Often, Random-Access Memory (RAM) is shared and equally accessible to all computing cores. This is called Uniform Memory Access (UMA) (Coffin 1992) or Parallel RAM (PRAM) (Casanova et al 2009) and allows for the exchange of information between different parallel programs.

In order to use parallel computing, the problem at hand needs to be parallelised. One approach is to partition the problem and to let each computing core work on one partition in parallel. When all partitions have been processed one instance of the problem has been solved. The other approach, which was taken in this work, is to have each computing core to process one instance of the problem in parallel. Since the processing time of one fitness evaluation might depend on the input data, it is advantages not to synchronise the parallel processes, because otherwise a fast process would waste computing time waiting for slower processes to catch up. This led to the development and successful application of Asynchronous Population-Based Hill Climbing (APBHC) (Nolle and Werner 2019), which is introduced in the next section.

ASYNCRONOUS POPULATION-BASED HILL CLIMBING

In Hill Climbing (HC) (Chandra and Hereendran 2014), which is a well-established optimisation strategy, a trial solution from the neighbourhood of a current solution is evaluated. If the new solution is better than the current one, the new solution becomes the current solution, otherwise the current solution remains the same. This will be repeated until a stopping criterion is fulfilled.

However, the performance of HC crucially depends on the definition of neighbourhood, i.e. the chosen step size (Nolle 2004): If the step size is too small, HC will eventually get trapped in the nearest local optimum. If a larger step size is used instead, the probability to escape local optima is increased, but at the same time, the chance of reaching the global optimum is decreased.

Communications of the ECMS, Volume 33, Issue 1,
Proceedings, ©ECMS Mauro Iacono, Francesco Palmieri,
Marco Gribaudo, Massimo Ficco (Editors)
ISBN: 978-3-937436-65-4/978-3-937436-66-1(CD) ISSN 2522-2414

This led to the development of adaptation schemes like Self-Adaptive Stepsize Search (SASS) (Nolle 2006), which is a population based Hill-Climbing algorithm that uses sampling to adjust the step size dynamically. This can be seen as collaboration between search entities, which are often called particles. In previous work, a parallel version of SASS, Asynchronous Population-Based Hill Climbing (APBHC) (Nolle and Werner 2017), was introduced. It allowed for n parallel, asynchronous threads to perform hill climbing, where n is the number of CPU cores available. The search strategy analysed in this work is a variation of APBHC and is explained below:

Initially, a random solution is generated and evaluated. Since at this point in time it is the only solution, it becomes the global best solution g. This global best solution is stored in shared memory, i.e. memory that can be read by all the threads simultaneously. Subsequently the n threads are started in parallel. Figure 1 shows the UML activity diagram for the main program.

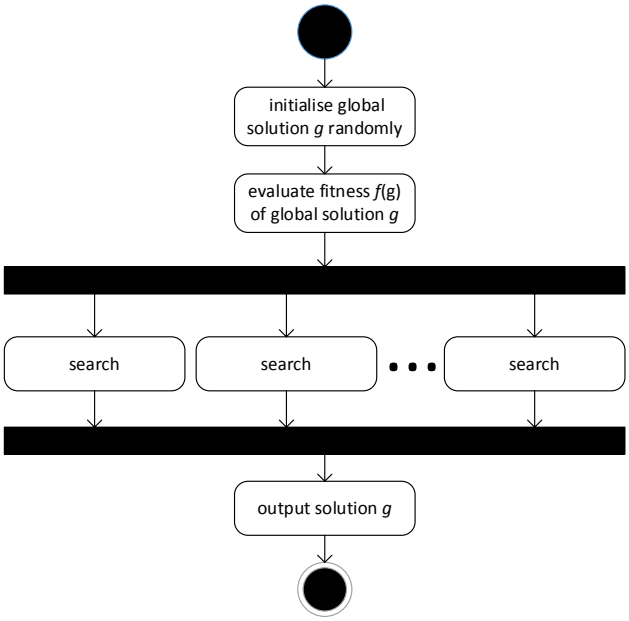

Figure 1: UML activity diagram for the main program.

If n is larger than the number of processing cores available on the computer system, multiple threads have to share a CPU core, potentially causing race conditions. This requires that the threads are non-blocking. Each thread executes then the search procedure depicted in Figure 2: A thread starts with a random current solution x. If x is the global best solution, a direct neighbour from x is chosen to become the new trial solution x'. Else, a new trial solution x' is generated by making a random step in the direction towards the global best g. If the fitness $f(x')$ is better than the fitness $f(x)$, x becomes x'. Otherwise, in contrast to APBHC, another trial solution x' is chosen randomly from the direct neighbourhood of x. If the fitness $f(x')$ is better than the fitness $f(x)$, x becomes x', otherwise x stays unchanged. Next, if a thread has found a better solution than the global best during its search, it updates the global best solution g. This process is repeated until a stopping condition holds. Once all the threads have finished their search, the global best solution g is returned as the overall solution.

Since the memory that stores the global best solution g is shared by all the threads, two or more threads might try to access this location at the same time. This potentially can result in data corruption. Hence it is necessary to protect the memory location whilst writing by setting a lock. Any other thread that tries to write has to wait until the lock is removed before it can update the global best solution.

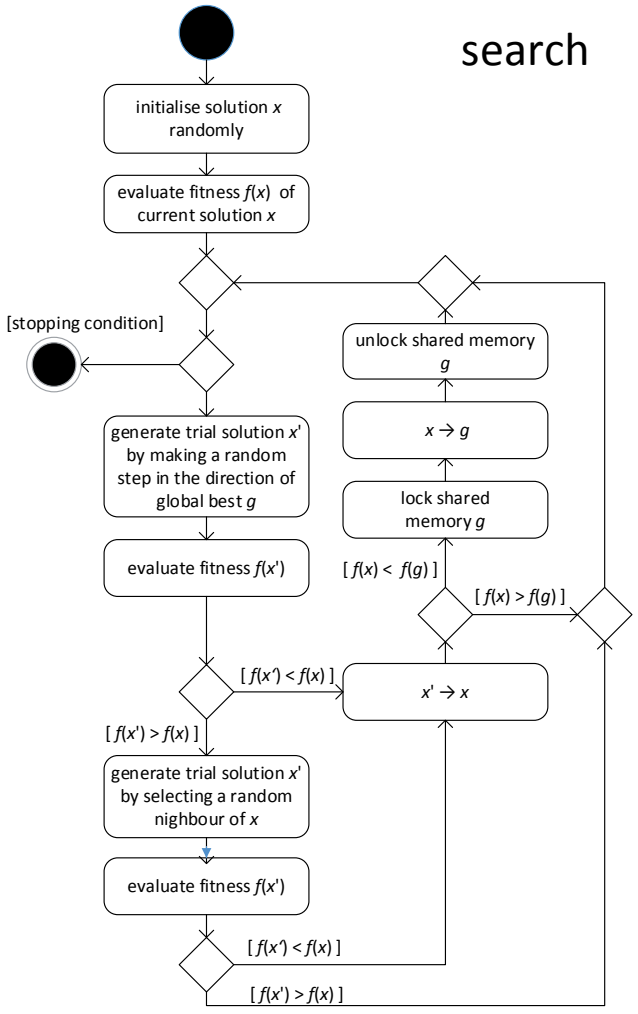

Figure 2: UML activity diagram showing the strategy of a single thread.

In the early stages of the search, the particles are randomly distributed across the search space and located near different local optima. Each particle performs Hill Climbing to continuously find better solutions in its vicinity. If a particle is not the current global best it means that there is at least one particle, which is located at a better position. Each particle now has a chance to

escape from its local optimum by trying a random step towards the current global best solution, otherwise it continues to decent into its own local minimum. This enables the population to explore the entire search space. Eventually, the particles home in onto the region containing the best local optimum found during the search, resulting in an exploitation of the most promising region of the search space.

In previous work, APBHC was successfully applied to a practical problem from the field of electric circuit design (Nolle and Werner 2019). This challenging problem is a discrete optimisation problem and has 9 design parameters. When plotting the error values of the global best solution *g* over time for different search runs, it was observed that each graph displayed two characteristic features. Figure 3 shows 10 new examples of this behaviour: All graphs exhibit two distinctive drops in the error value. Besides those drops, the graphs show a rather flat decline in error values over time.

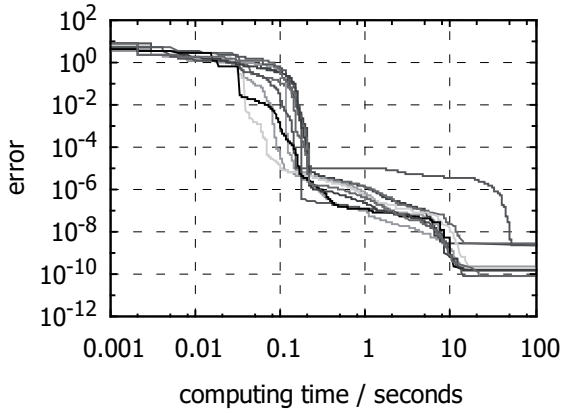

Figure 3: Error values for 10 exemplary search runs.

In order to get more insight into the dynamics of the algorithm, an analysis of its internal behaviour has been carried out and is presented below.

ANALYSIS

A common way of evaluating the performance of search strategies is to plot the development of the error function value over time. As mentioned above, when plotting the error values for different search runs, it was observed that each graph displayed two characteristic features. Figure 4 shows the development of the error value over time for a single exemplary search run where the same control parameter settings were used as in (Nolle and Werner 2019).

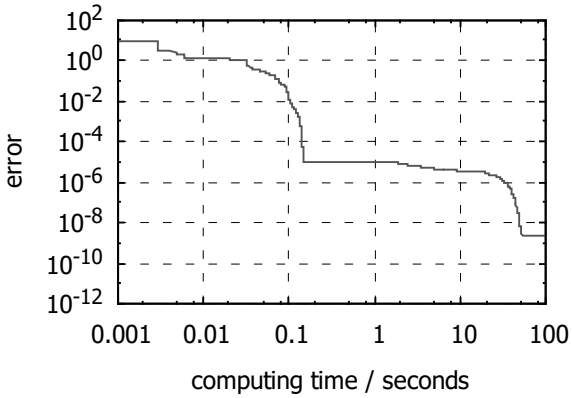

Figure 4: Error value over time for a single typical search run in logarithmic scale.

At the beginning of the search, the decrease of the error value is relatively small. After approximately 0.03 seconds, the start of a steep drop can be observed, which lasts until approximately 0.15 seconds. The next significant change in behaviour begins at approximately 11 seconds. After 54 seconds, the error value remains almost the same. A similar behaviour could be observed in each of the 10 simulations, which were carried out (Figure 3).

In order to understand these features, the inner workings of the algorithm were monitored: the number of jumps out of a local optima was counted and logged over time. The search threads are working in parallel and asynchronously. That means that some threads may have to share a computing core and therefore run slower than others. Therefore, it was not possible to let the search threads to report about their number of jumps, say, after a fixed number of iterations or fitness evaluations, because different threads might take different lengths of time for the same number of iterations. Instead, an additional monitor thread was introduced and started simultaneously with the search threads (Figure 5).

Each search thread increments a shared variable every time when jumping out of the local optimum. This shared variable has to be protected from being accessed simultaneously by more than one thread by locking and unlocking it by a thread when access is granted. The monitor thread waits for 10 milliseconds. It then reads out the total count of jumps for all the threads and writes it into a log file. It then checks if there are still any active search threads. If so, it loops back, otherwise it finishes (Figure 6).

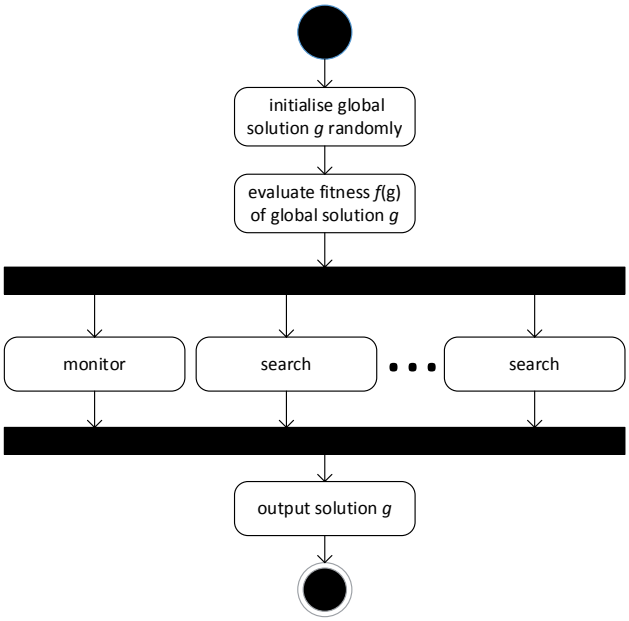

Figure 5: UML activity diagram with additional monitor.

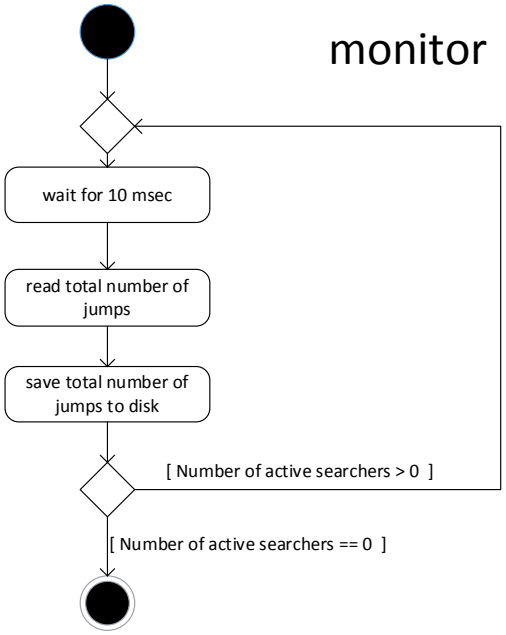

Figure 6: UML activity diagram of the monitor thread.

The number of active search threads is counted in a shared variable. On start, each search thread is incrementing this variable, and on finishing it decrements it. Hence, this variable always holds the current count of active searchers. This setup was then used to examine the real-time behaviour of the search algorithm, which is discussed in the next section.

DISCUSSION

The logged data from a typical search run is plotted in Figure 7. It shows the total number of jumps over time. In addition, the time derivative, i.e. jumps per computing time, is presented in Figure 8 for the same exemplary run. From these graphs it can be seen that the significant reduction of the error function starting after 0.03 seconds is in good agreement with the strong rise of the jump rate (from around 15 to more than 100 jumps per millisecond). During this period of time (approximately 0.14 seconds) the error function declines by a factor of 10^5. In the subsequent time window (up to 1 second) the jump rate stays at around zero jumps per millisecond.

After about 1 second computing time, a burst in the jump rate can be observed, which lasts for about 9 seconds. It then drops back below 10 jumps per millisecond until it is followed by another burst, lasting for about 33 seconds. During this burst, another steep decline in the error value can be observed (Figure 4). After this, no significant improvement of the error values takes place. This correlation of the jump rate and the error convergence was observed in all simulations that were carried out.

This observation supports the assumption made above that at the beginning of a search run the particles carry out an exploration of the search space. They then go over into exploitation. Yet, they still have the ability to escape their local optima as it can be seen from the sudden burst in the jump rate (Figure 8). Each jump indicates that a particle has jumped to a better solution, which means into a more promising region of the search space. It later exploits this region via Hill Climbing. Toward the end of the search, all the particles are located in the most promising region and therefore no more jumps out of that region can be observed.

Nevertheless, based on this finding, it is not recommended to use the jump rate as a stopping criteria, because, as seen in Figure 8, sudden rises in the jump rate might occur, allowing for better exploration of the search space and hence may result in better solutions.

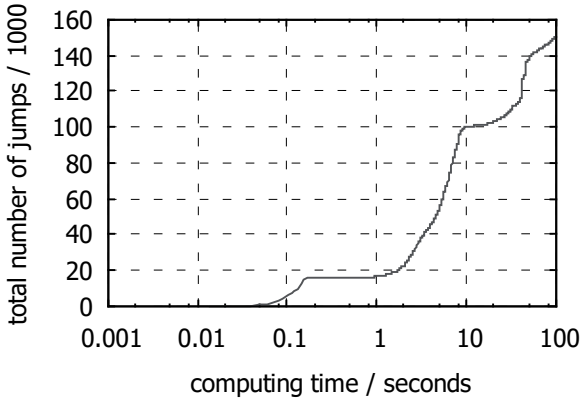

Figure 7: Total number of jumps over computing time for the exemplary search run in Figure 4.

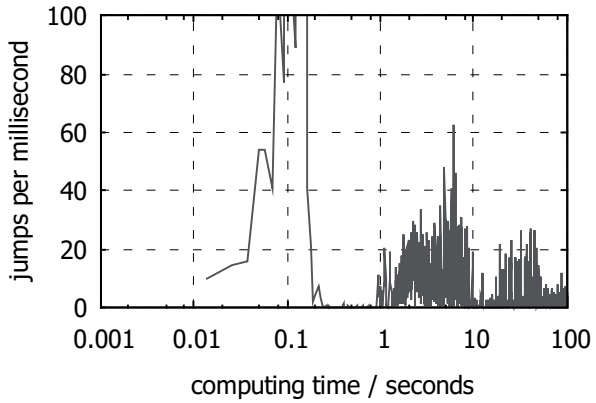

Figure 8: Jump rate over computing time for the exemplary search run in Figure 3.

CONCLUSIONS

In previous work, a variation of Asynchronous Population Based Hill Climbing (APBHC) was successfully applied to a discrete optimisation task from the field of electronic circuit design. In this work, the characteristic behaviour of APBHC observed previously was analysed. A challenge here was the asynchronous nature of the search algorithm, which did not allow the search threads to save the internal state data into log files because they took different lengths of time. Instead, a new monitoring strategy was developed. Here, a separate monitor thread reads accesses internal data of the search threads via shared memory in specified time intervals until no more search thread is present. It was shown that APBHC is well capable of escaping local optima whilst still using the local search capability of Hill Climbing. A strong correlation between the error convergence and the jump rate could be observed. Therefore, APBHC is able of switching between exploration and exploitation of the search space adaptively during a search. The online monitoring of the jump rate made this observation possible.

REFERENCES

Casanova, H., Legrad, A., Robert, Y. (2009) *Parallel Algorithms*, CRC Press.

Chandra, S.S.V. and Hereendran, S. A. (2014) *Artificial Intelligence and Machine Learning*, PHI Learning.

Coffin, M. H. (1992) *Parallel Programming a New Approach*, Prentice Hall.

Dorigo, M., Gambardella, L. (1997) "Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem". *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp 53-66.

Emmerich M., Giotis A., Özdemir M., Bäck T., Giannakoglou K. (2002) "Metamodel - Assisted Evolution Strategies", *Lecture Notes in Computer Science*, Vol 2439, Springer.

Flynn, M. J. (1972) "Some computer organizations and their effectiveness", *IEEE Transactions on Computers*, Vol. 21, Issue 9, pp 948-960.

Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization*, and Machine Learning, Addison-Wesley.

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press.

Kennedy, J., Eberhart, R. (1995) "Particle swarm optimization", Proceedings of IEEE International Conference on Neural Networks, Vol.4, pp 1942-1948.

Kirkpatric, S., Gelatt, C. D., Vecchi M. P. (1984) "Optimization by Simulated Annealing: Quantitative Study", *Journal of Statistical Physics*, Vol.34, pp 975-986.

Nolle, L. (2004) "On the effect of step width selection schemes on the performance of stochastic local search strategies".

Nolle, L. (2006) "On a hill-climbing algorithm with adaptive stepsize: Towards a control parameter-less black-box optimisation algorithm," *Advances in Soft Computing*, Vol. 38, 2006, pp. 587–595.

Nolle, L., Krause, R., Cant, R.J. (2016) "On Practical Automated Engineering Design". in: Al-Begain, K., Bargiela, A. (Eds): *Seminal Contributions to Modelling and Simulation*, Springer.

Nolle, L. and Werner, J. (2017) "Asynchronous population-based hill climbing applied to SPICE model generation from EM simulation data", *Lecture Notes in Computer Science*, Vol. 10630. Springer, 2017, pp. 423–428.

Nolle, L., Werner, J. (2019) "SPICE Model Tuning using a Parallel Computational Optimisation Strategy", to appear in: Proceedings of 2019 Joint International Symposium on Electromagnetic Compatibility and Asia-Pacific International Symposium on Electromagnetic Compatibility, Sapporo, Japan, June 3-7, 2019.

AUTHOR BIOGRAPHIES

LARS NOLLE graduated from the University of Applied Science and Arts in Hanover, Germany, with a degree in Computer Science and Electronics. He obtained a PgD in Software and Systems Security and an MSc in Software Engineering from the University of Oxford as well as an MSc in Computing and a PhD in Applied Computational Intelligence from The Open University. He worked in the software industry before joining The Open University as a Research Fellow. He later became a Senior Lecturer in Computing at Nottingham Trent University and is now a Professor of Applied Computer Science and Dean of Studies at Jade University of Applied Sciences. His main research interests are computational optimisation methods for real-world scientific and engineering applications.

JENS WERNER was born in Cologne, Germany in 1969. He received his Dipl.-Ing. and Dr.-Ing. Degrees in Electrical Engineering from the Technical University of Braunschweig, Germany, in 1996 and 2002, respectively. In 1996 he was working with Aerodata AG as a Flight Inspection Engineer on calibration of airborne antennas. From 1996 to 2001, he was a Research Assistant at the Institute of Electromagnetic Compatibility, TU Braunschweig. His main research interests were measurement techniques and representation of guided and radiated electromagnetic fields. In 2001 he joined the Innovation Centre of Philips Semiconductors Germany GmbH in Hamburg, (since 2006 NXP Semiconductors). In March 2014, he became a Professor at Jade University of Applied Sciences, Wilhelmshaven, Germany. He is responsible for the RF, Wireless and EMC laboratory.