# HC12: Highly Scalable Optimisation Algorithm

Radomil Matousek

**Abstract.** In engineering as well as in non-engineering areas, numerous optimisation problems have to be solved using a wide range of optimisation methods. Soft-computing optimisation procedures are often applied to problems for which the classic mathematical optimisation approaches do not yield satisfactory results. In this paper we present a relatively new optimisation algorithm denoted as HC12 and demonstrate its possible parallel implementation. The paper aims to show that HC12 is highly scalable and can be implemented in a cluster of computers. As a practical consequence, the high scalability substantially reduces the computing time of optimisation problems.

## 1 Introduction

This paper describes the possibility of parallelizing the HC12 optimisation algorithm. Designed in 1995 [3], HC12 algorithm was well described e.g. in [4, 5]. This original algorithm uses a hill-climbing approach [6], more precisely: for a given optimisation problem, in each iteration step $i$, a solution $(\mathbf{A}_{kernel,i})$ exists to which a neighbourhood of further possible solutions is generated using a fixed pattern. From this neighbourhood, a best solution is chosen for iteration step $i+1$, which will again be used to generate a new solution $(\mathbf{A}_{kernel,i+1})$. The algorithm stops if no best solution can be found, that is, if (for a minimisation problem)

$$\min\left(f(\mathbf{A}_{\text{kernel,i}})\right) \leq \min\left(f(\mathbf{A}_{\text{kernel,i+1}})\right), \tag{1}$$

where $i$ is the iteration number and $f$ is the objective function. As Chapter 2 and Chapter 4 shows, the HC12 algorithm is, among others, designed to lend itself to

Radomil Matousek
Brno University of Technology, Faculty of Mechanical Engineering,
Department of Applied Computer Science,
Technická 2, Brno 616 69, Czech Republic
e-mail: matousek@fme.vutbr.cz

J.R. González et al. (Eds.): NICSO 2010, SCI 284, pp. 177–183, 2010.
springerlink.com 

good parallelization. Thus the paper aims to verify this property by implementing the algorithm in a cluster of computers. The test has been designed to verify and demonstrate a computation-time reduction in problems for which "this is already significant" in terms of time. As a testing problem, the F6 optimisation problem has been chosen for which, given a complexity and the parameters of the SW and HW environment, an average single-processor computing time of 45 minutes has been achieved.

## 2 HC12 Algorithm

A mathematical description of the algorithm can be found in [5]. Here the basic ideas are summarized:

- The solution of a given optimisation problem is represented by a binary vector $A$.
- This binary vector $\mathbf{A}$ codes $k$ real parameters of the optimisation problem, that is, the real input parameters $x_i$ (where $i \in \{1...k\}$) of the objective function. This provides a basis for discretizing the Domain of Definition (DoD) of the problem parameters to be found. The degree of discretization depends on the size of the binary string being proportional to $2^s$ where $s$ is the number of bits per parameter.
- The procedure used to decode the binary string is given by the way the optimisation problem is formulated. For F6, the binary vector $\mathbf{A}$ is decoded using the Gray coding on a vector of integer parameters $(int_1, ..., int_k)$ where $k = 50$ in our case. Next, using the DoD, this vector is further transformed into a vector of real parameters $x_i$.
- In the first iteration, a binary vector $\mathbf{A}_{kernel,1}$ and a neighbourhood to fit a fixed pattern are randomly generated. With HC12, this is a neighbourhood with distances 1 and 2 from vector $\mathbf{A}_{kernel}$ in the sense of the Hamming metric. In each iteration, the best solution is chosen as the new basis. The procedure is repeated until condition (1) is satisfied.

The Hamming distance $\rho_H$ between two binary vectors of equal length is the number of positions for which the corresponding symbols are different. Let $\mathbf{a}, \mathbf{b}$ are binary vectors of length $N$ and $a, b$ its elements, then the Hamming distance can be calculated as follows:

$$\rho_H(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{N} |a_i - b_i| \tag{2}$$

The principle of decoding a binary string to a vector of real parameters is shown in Fig. 1, the generating of a neighbourhood using a four-bit vector $\mathbf{A}_{kernel}$, is shown in Fig. 2. It follows from the principle that the cardinality (size) of the neighbourhood for a Hamming distance of 1 corresponds to the length $N$ of the binary vector thus growing linearly with the length of the binary vector. On the other hand, the cardinality (size) of a neighbourhood for a Hamming distance of 2 corresponds to the combination number $(N, 2)$. It is exactly this type of neighbourhood that causes an unwelcome combinatorial expansion of the neighbourhood generated whose size grows exponentially with the length of the binary vector.

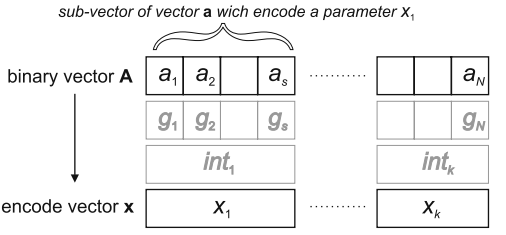


**Fig. 1** Parameters' encoding scheme (binary and Gray code, integer and real parameters)

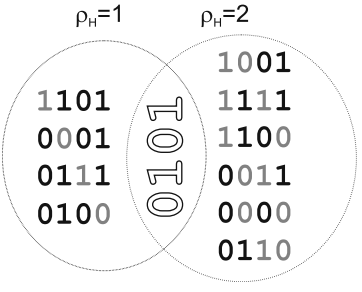


**Fig. 2** An example of 4-bits neighbourhood generating for $\rho_H = 1$ and $\rho_H = 2$

## 3 The F6 Test Function

To demonstrate the performance of the HC12 algorithm and to verify its scalability in a cluster of computers, a multimodal function has been used which is usually used to test the power of both standard and soft-computing optimisation algorithms referred to as the Rastrigino F6 function.

$$\begin{aligned} &F_6(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right) \\ &-5.12 \leqslant x_i \leqslant -5.12,\ \min F_6(\mathbf{x}) = F_6(0,...,0) = 0 \end{aligned} \tag{3}$$

As follows from definition (3), this function is continuous and smooth, but has a large number of maxima and minima. This number grows as a power given by the dimension of the function. In the present example, the optimisation problem is solved for 50 variables with an identical domain of [-5.12, 5.12]. The discretization chosen yields a calculation precision of $eps = 0.01$ for each parameter $x_i$.

It should be stressed that, for a minimisation problem, the F6 function represents 1.1739e+052 possible extremes on a given domain. Moreover, each of the variables to be found is encoded using 10 bits, which, given the number of parameters, results in a 500-bit binary string. If a rude-force algorithm were used to find an optimum solution, such a length would require 3.2734e+150 possible variants! This clearly demonstrates the enormous time needed to solve such an optimisation problem. As the first iteration of the HC12 algorithm is of stochastic nature, all the tests have been

made for 100 algorithm runs with the average taken as a result since the median and average are very close.

To be able to analyse the scalability, for each configuration (the number of computers clustered), we used the same "random" vector configurations of the first iteration $\mathbf{A}_{kernel,1}$ The below table shows an example of the results of a given optimisation problem.

**Table 1** Record of the optimisation process of looking for a minimum of the F6 function

| Iteration | Objective Function Value |
|---|---|
| 1 | 884.3852097349522 |
| 2 | 805.3840740133274 |
| 3 | 730.8040415793919 |
| ... | ... |
| 30 | 121.2632026930635 |
| ... | ... |
| 60 | 21.1934433716936 |
| ... | ... |
| 100 | 0.6542459906284 |
| ... | ... |
| 122 | 0.0000000000000 |

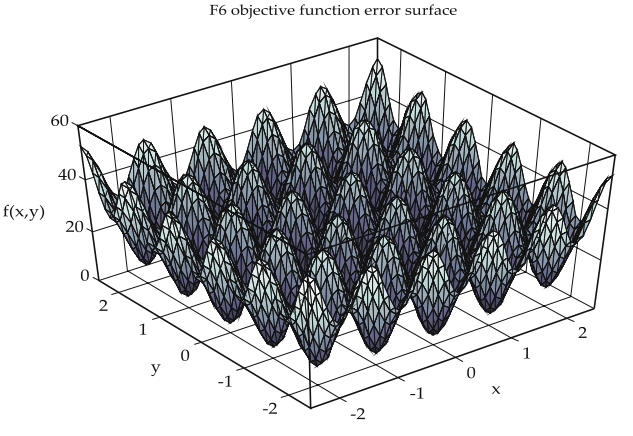


**Fig. 3** Visualization of Rastrigin's function F6 in the range (DoD) from -5 to 5. The global optimum is positioned at point [0, 0]

## 4 Parallel Implementation

The optimization method was implemented as parallel application in Java and deployed on a cluster of computers.

The computers had AMD Opteron CPUs running at 2.6GHz and Linux operating system, the Java platform used was SUN 64-bit JDK version `1.6.0_14` with the Server VM.

The application used a master-slave architecture, where slaves registered with the master and the master then assigned them chunks of work. The master and slave parts communicated using the RMI (Remote Method Invocation) mechanism provided by the Java platform.

The master part coordinated repeated iterations, where each iteration involved:

- taking the so far best known $N$-bit vector as a starting vector;
- finding all $N$-bit vectors with Hamming distance of 1 from the starting vector - there are just $N$ of them, so parallelisation is not needed;
- converting all of them into vectors of floating point numbers with double precision, computing the value of the objective function and finding the minimum value, eventually replacing the starting vector with the new minimum;
- dividing the $N(N-1)/2$ $N$-bit vectors with Hamming distance of 2 from the starting vector into chunks of equal size and assigning them to slaves to compute the value of the objective function for each of them, returning the best one found;
- collecting the best found bit vectors from slaves and selecting the best of them;
- if the best found bit vector is better than the starting point, it uses it as a starting point for the next iteration, otherwise it ends by equation (1).

This list shows that the master part makes synchronization points at the end of each iteration, so the Amdahl's law predicts that the speedup cannot be fully linear with growing number of CPUs. But for growing number of bits in vectors, the number of vectors processed by the parallel part of the algorithm is growing faster compared to the number of vectors processed by the serial part, and the maximum possible speedup is improving.

## 5 Results and Conclusions

The results of a parallel implementation of the optimization process for 500 bits, 50 dimensions by 10 bits each, and objective function F6 are presented by Table 2 and Fig. 3. An average count of iterations to reach the best HC12 solution was 122 (an example of iteration run is in the Table 1). A code was found to scale nearly linearly on a cluster of computing machines if each machine runs just one instance of the slave part. However, when more than one slave was located on the same machine, the scalability was much worse. For example, a machine with 8 dual-core AMD Opteron CPUs was tried with 1, 2, 4, 8, and 16 instances of the slave part. When using 8 slaves, the computation was only about 4 times faster than when using 1 slave. When using 16 slaves (the machine had 16 CPU cores so that improvement was possible) the computation was even slower than when using only 8 slaves. This

**Table 2** Scalability vs. computing time

| CPU[a] | time | speed up |
|---|---|---|
| 1 | 2722 [s] | 1.00 x |
| 2 | 1400 [s] | 1.94 x |
| 3 | 948 [s] | 2.87 x |
| 4 | 697 [s] | 3.91 x |
| 5 | 571 [s] | 4.77 x |
| 6 | 479 [s] | 5.69 x |
| 7 | 437 [s] | 6.23 x |
| 8 | 382 [s] | 7.13 x |
| 9 | 336 [s] | 8.10 x |

[a] The number of CPUs in cluster (AMD Opteron, two cores).

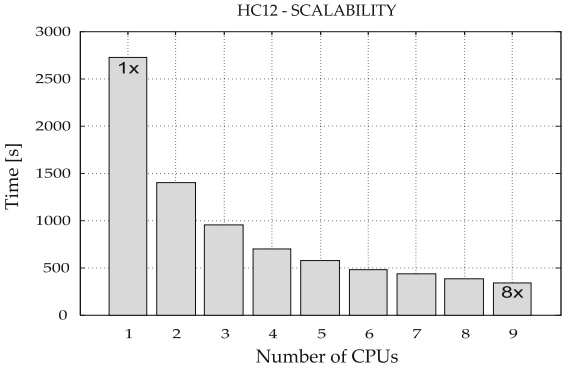


**Fig. 4** Computing time depending on the number of CPUs in the cluster of computers

result is likely to be caused by scalability issues of multiprocessor machines, and does not provide any information about the scalability of the optimization method itself.

The testing of a parallel implementation if the HC12 algorithm has proved unequivocally that the algorithm is highly scalable within a cluster of computers. For a given optimisation problem, the difference between the computing time on a single computer (about 45 minutes) and 9 computers (about 5 minutes) is about 40 minutes as given by the measurements (see Table 2). This acceleration may be considered significant. As the favourable properties of the HC12 optimisation algorithm or its implementations within other soft-computing methods (such as GAHC algorithm [5]) have already been demonstrated several times for a single CPU, the authors will further focus on the research and applications of its parallel implementations. In this light, this paper demonstrating a high scalability of the HC12 algorithm may be thought of as one of the pilot papers concerning parallel implementations of the HC12 algorithm.

**Acknowledgements.** The access to the MetaCentrum supercomputing facilities provided under the research intent MSM6383917201 is highly appreciated. This work was supported by the Czech Ministry of Education in the frame of MSM 0021630529 "Intelligent Systems in Automation" and by the Grant Agency of the Czech Republic No.: 102/091668 "Control Algorithm Design by means of Evolutionary Approach".

## References

[1] Amdahl, G.: Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In: AFIPS Conference Proceedings, vol. (30), pp. 483–485 (1967)

[2] Battiti, R., Tecchiolli, G.: Local search with memory: Benchmarking RTS. journal of Operations Research Spectrum 17(2/3), 67–86 (1995)

[3] Matousek, R.: GA (GA with HC mutation) – implementation and application, Master thesis (in Czech), Brno University of Technology, Brno, Czech Republic (1995)

[4] Matousek, R.: GAHC: A Hybrid Genetic Algorithm. In: Proceedings of the 10th Fuzzy Colloquium in Zittau, Zittau, pp. 239–244 (2002) ISBN: 3-9808089-2-0

[5] Matousek, R.: GAHC: Improved Genetic Algorithm. In: Krasnogor, et al. (eds.) Nature Inspired Cooperative Strategies for Optimization (NICSO 2007). Springer book series, vol. 129, XIV, p. 114 (12p). Springer, Berlin (2008)

[6] Mitchell, M., Holland, J.H.: When Will a Genetic Algorithm Outperform Hill Climbing? In: Cowan, J.D., Tesauro, G., Alspector, J. (eds.) Advances in Neural Information Processing Systems, vol. 6. Morgan Kaufmann, San Mateo (1994)

[7] Zhou, R., Hansen, E.A.: Breadth-First Heuristic Search. In: 14th International Conference on Automated Planning and Scheduling (ICAPS 2004), Whister, British Columbia, Canada (2004)