Parallel Gene Transfer Operations for the Bacterial Evolutionary Algorithm

M. Hatwagner, A. Horvath

SZE, 9026 Győr, Egyetem tér 1.
Phone: 503 400, fax: 503 400
e–mail: miklos.hatwagner@sze.hu, horvatha@sze.hu

Abstract: Bacterial evolutionary algorithm (BEA) is a special evolutionary method, originally developed to optimize fuzzy system parameters, but it is also useful in many design and engineering problems. In the latter case, applications kill most of their time with the evaluation of the objective function. One possible way to speed up BEA is the use of many CPUs, and evaluate the objective functions of different individuals on them simultaneously. This can be realized e.g. with a cluster of workstations or with an SMP/SMC system. Unfortunately, the BEA in its original form is not suitable for parallelization, because one of the two main operations, the "gene transfer" is sequential. In this article we propose three new alternatives of BEA and present the results of test calculations with 5 test problems, on 1 to 16 processors to find a good parallel version of BEA.

Keywords: Bacterial Evolutionary Algorithm, parallel computing.

1. Introduction

Evolutionary algorithms are optimisation methods inspired by the Darwinian theory of evolution. The four main tends are evolutionary strategies, evolutionary programming, genetic programming and genetic algorithms (GA) [1]. The bacterial evolutionary algorithm (BEA) is a relatively new member of evolutionary algorithms, proposed by Norberto Eiji Nawa and Takeshi Furuhashi [13][14][15]. It is a descendant of the Pseudo-Bacterial Genetic Algorithm (PBGA) and traditional GA [5].

Several researchers have been dealing with the usage of memetic algorithms [6][11][14] to make evolutionary algorithms more efficient. The first implementation of them related to BEA was the Bacterial Memetic Algorithm (BMA) [2]. The simple genetic operators of Microbial Genetic Algorithm (MGA) [7][8] were successfully used in BMA as well to solve the travelling salesman problem [4].

BEA is used primarily to discover or optimize fuzzy system parameters, but it is very useful in many other cases. It can be considered as a global search algorithm, where an exact answer is not always required; thus, a near optimal solution is acceptable. It is able to solve and quasi-optimize complex optimization and design problems having non-linear, high-dimensional, multi-modal, and non-continuous characteristics. This algorithm does not demand the use or the existence of derivatives of the objective functions, such as in the case of gradient-based methods.

BEA always keeps a record of the possible solutions. The solutions are also called bacteria, they form together the population. BEA includes two operations: the so-called “bacterial mutation”, and the “gene transfer” operation of the PBGA. Bacterial mutation optimizes the bacteria individually. The recombination operation of the GA was replaced by gene transfer in PBGA and BEA. This operation allows bacteria to directly transfer information to other bacteria of the population (see Figure 1).

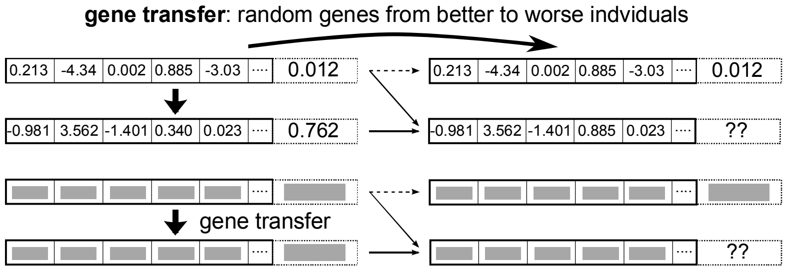

Figure 1. Gene transfer allows each bacterium to directly transfer genetic information into another bacterium.

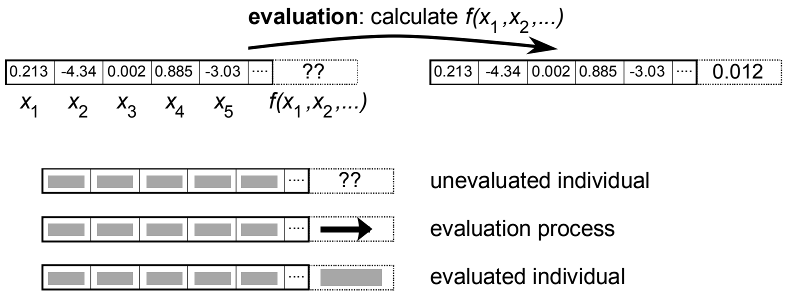

Figure 2. The destination bacterium has to be re-evaluated after the gene transfer.

Using these two operators (mutation and gene transfer) a new population can be created. A series of populations leads to the final population, when some kind of termination condition is fulfilled. The bacterium with the best objective function value will be accepted as the solution of the original problem. The exact details of the algorithm can be found in [13].

A significant number of real applications kill most of their time with the evaluation of the objective functions (see Figure 2), because both of the operations mentioned above use the objective values frequently. In a complex optimization problem evaluation of the objective function may require a complete finite element calculation or solving a system of ordinary differential equations. (See e.g. [9].) In such cases the cost of an objective function evaluation is much higher than the one of starting a new process or communicating with another machine. Thus, if the optimization problem is a complex one, simultaneous evaluation of the objective function for different individuals can save a lot of time.

There is a trivial way of making the bacterial mutation parallel, because all the bacteria can be mutated at the same time. All clones of a specified bacterium can be evaluated simultaneously as well, but only one gene can be modified in each computational cycle, i.e. several genes cannot be modified parallel. If the population consists of N_p bacteria, the number of clones is N_c and every bacterium contains N_g genes, the mutation operation evaluates the objective function $E_o = N_p N_c N_g$ times.

With $N_{cpu} = N_p N_c$ processors it is possible to calculate the objective values in N_g computational rounds, thus the speed-up of the calculation can be theoretically even $S_m = \frac{E_o}{N_g} = N_p N_c$ [10]. (It is assumed in the article, that the time of objective function evaluation is constant and independent of the content of the chromosome.) In most cases N_{cpu} is much bigger then the number of available CPUs in today's systems, so it can be declared, that bacterial mutation in its original form is suitable to run on a multicore/multiprocessor system. (In a typical calculation $N_p \approx 30-100$, $N_c \approx 20-50$, therefore $N_{cpu,\max} \approx 600-5000$.)

The second main operation of the BEA, the gene transfer sorts the bacteria based on their objective values. After then, it divides the population in two halves. The bacteria with better objective values get into the superior half, the others in the inferior half. After then, the algorithm selects randomly the source bacterium from the superior half, and the destination bacterium from the inferior half. Some selected parts of the source bacterium will be transferred to the destination bacterium, so the objective value of the modified bacterium has to be recalculated. The whole process starting from the sorting of the population is repeated several (N_{gt}) times (see Figure 3 and legend in Figure 2).

It is obvious, that the newly evaluated bacterium may belong to any half of the population, according to its objective value and can cause an originally superior individual to become inferior. Thus the successive gene transfers are not independent, so it is impossible to evaluate the objective value of more bacteria at the same time in the original form of BEA. The only way of making the gene transfer operator parallel, is to create an alternative, new version of it. In the next section, the authors propose a variant of the original BEA, which permits the parallel evaluation of new individuals. Furthermore we examine a variant of the MGA, which is similar to BEA but can be made parallel at a limited degree. Note the parallel nature of MGA is mentioned in [4], however here we show an implementation which guaranties the parallel evaluation in the case of arbitrary number of processors. Finally we show a 4th version, a fully parallel MGA-variant.

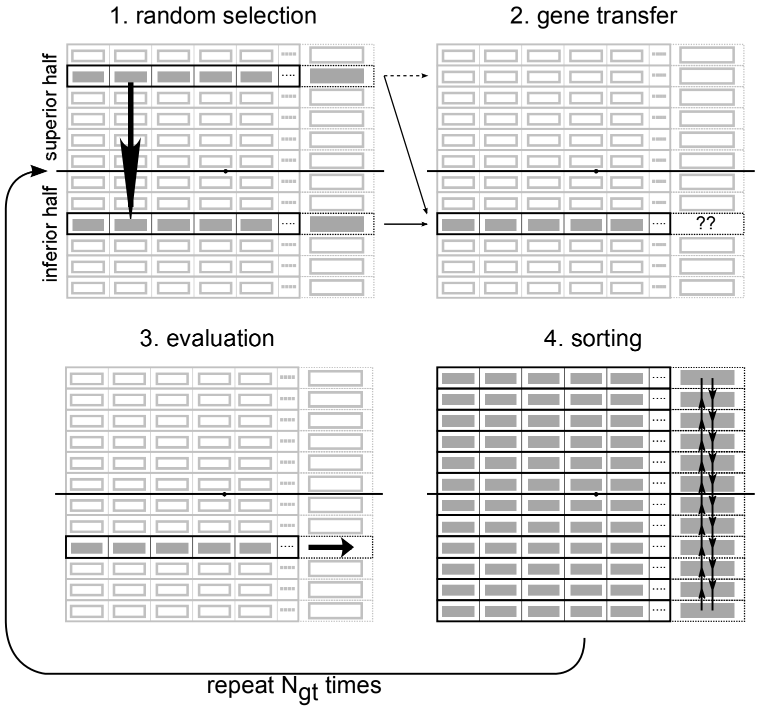

Figure 3. Operation of the original gene transfer

2. New kinds of gene transfer operators

2.1. Original gene transfer with auxiliary population

In this version the sorting of the population on the grounds of the objective values and selection of source and destination bacteria from inferior and superior half are the same as before. The main idea of this version is to keep the population untouched and collect the modified (infected) individuals to an "auxiliary population", and start a parallel evaluation of these individuals only when this auxiliary population is full. After this evaluation, the original and auxiliary populations are sorted together and the best N_p will be the new population, and the worse N_{aux} will be deleted. This has to be repeated until N_{gt} gene transfers are executed.

The size of the auxiliary population (N_{aux}) is a free parameter of the algorithm, but practically it is a multiple of the number of available processors.

The operation can be described with the following C-based pseudo code. The same algorithm is visualized in Figure 4.

```
/*
   gt: the remaining number of gene transfers
   numberOfGTs: the total number of gene transfers per generation
   auxPopSize: size of the auxiliary population
*/
for(gt=numberOfGTs; gt>0; gt-=auxPopSize) {
   sort(population);
   numberOfNewBacteria = gt>auxPopSize?auxPopSize:gt;
   for(auxIndex=0; auxIndex<numberOfNewBacteria; auxIndex++) {
      src = rnd(0, popSize/2);  /* rnd(a, b) returns an integer in the range [a; b) */
      dest = rnd(popSize/2, popSize);
      newBact = population[dest];
      gene = rnd(0, numberOfGenes);
      newBact[gene] = src[gene];  /* transferring a gene from source to destination */
      auxPop[auxIndex] = newBact;
   }
   evaluateObjectiveFnsParallel(auxPop);
   sort(auxPop);
   /* copy the best popSize pieces of bacteria from population and auxPop to population
*/
   merge(population, auxPop, popSize);
}
```

An interesting property of the original gene transfer is that the objective value of the infected destination bacterium can be worse, than the former value. As opposed to this, using auxiliary population only better bacteria can replace old ones. It can be considered as a kind of elitism. However, there is a big danger in using the auxiliary population: it may make the genetic diversity worse. It may happen that the profit of parallel evaluation will be less than the deficit of this effect. Therefore the performance will be tested in several cases to measure the whole behaviour of this new version.

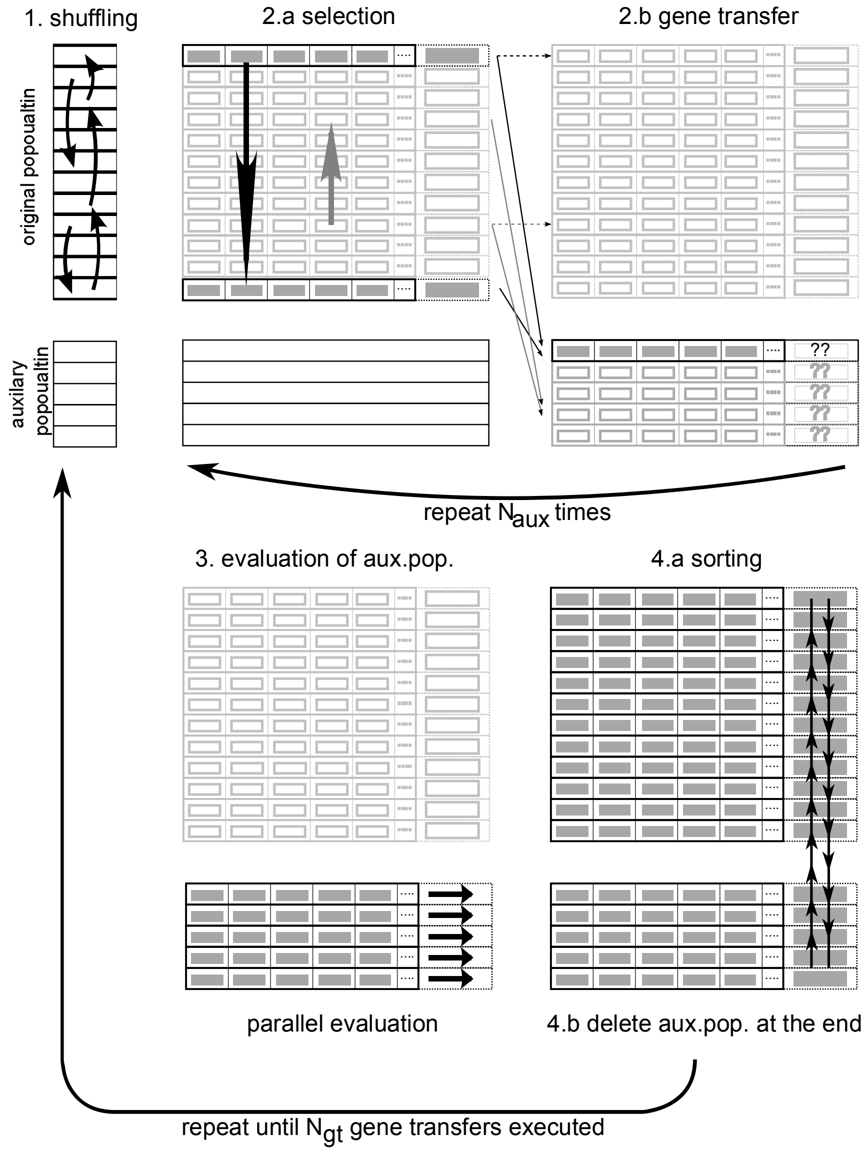

Figure 4. Operation of the original gene transfer with auxiliary population

2.2. Gene transfer inspired by Microbial Genetic Algorithm

The goal of creating the Microbial Genetic Algorithm (MGA) was to strip down the original GA and reduce it to its basics. In one cycle the MGA picks two individuals at random from the population. The one with the better objective function is called winner, the other is the loser. During recombination the winner infects the loser with some portion of its genetic material. The recombination is followed by mutation and the replacement of the loser with its genetically modified chromosome in the population.

The MGA gave the idea of creating a new gene transfer operation (hereafter this operation is signed shortly as MGA in figures and tables), because its selection and recombination can be regarded as a gene transfer operation. It is necessary to create disjoint source-destination pairs for the sake of parallel execution of objective functions. The number of such pairs cannot be more than half of the population size at a time. If more gene transfers are needed, the process has to be repeated.

The following pseudo-code and Figure 5 shows the MGA-based gene transfer method.

```
src = popSize;  /*index of source bacteria */
dest = src-1;  /* index of destination bacteria */
/* numberOfGTs: the total number of gene transfers per generation */
for(gt=0; gt<numberOfGTs; gt++) {
   if(src>=dest) {
      /* evaluate popSize-dest-1 pieces of bacteria from dest+1 */
      evaluateObjectiveFnsParallel(population, dest+1, popSize-dest-1);
      src = 0;
      dest = popSize-1;
      shuffle(population);
   }
   /* move the loosers back (lower objective value is better) */
   if(objectiveValue(population[src]) > objectiveValue(population[dest])) {
      commute(population[src], population[dest]);
   }
   gene = rnd(0, numberOfGenes);  /* rnd(a, b) returns an integer in the range [a; b) */
   /* transferring a gene from source to destination */
   population[dest][gene] = population[src][gene];
   src++;
   dest--;
}
/* evaluate the remainders */
evaluateObjectiveFnsParallel(population, dest+1, popSize-dest-1);
```

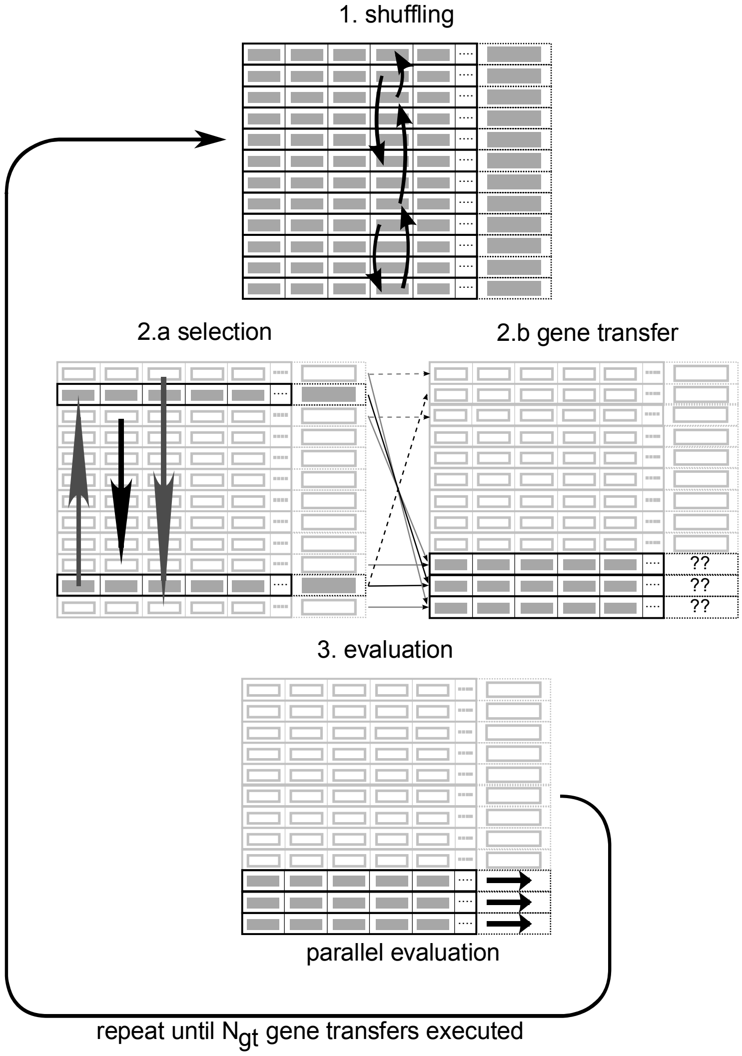

Figure 5. Operation of the gene transfer inspired by Microbial Genetic Algorithm

The "shuffling" step is a simple algorithmic trick: after randomization of positions in the population, any fixed pair-making strategy will produce random, disjoint pairs. We choose an inward advancing coupling strategy: the first pair will contain the 1^{st} and $N_p{}^{th}$ element (from shuffled population), the second pair will contain the 2^{nd} and $N_p - 1^{th}$ element, and so on.

2.3. Gene transfer inspired by MGA with auxiliary population

The MGA inspired gene transfer can be extended with an auxiliary population. In this case the $N_p / 2$ limit for the number of parallel evaluations can be exceeded.

This algorithm picks two bacteria at random. A new bacterium will be created in the auxiliary population, using the previously selected two bacteria's genetic information. The new bacterium inherits the genes of the destination bacteria, except one, which comes from the source. The auxiliary population can be filled with repeated infections, and all the bacteria of the auxiliary population can be evaluated simultaneously. The original and auxiliary populations have to be merged and the N_{aux} least fit bacteria have to be dropped.

This method is elitist, and the number of parallel evaluations is not limited to half of the population size.

The following pseudo-code and Figure 6 shows the MGA-based gene transfer method with auxiliary population.

```
/*
  gt: the remaining number of gene transfers
  numberOfGTs: the total number of gene transfers per generation
  auxPopSize: size of the auxiliary population
*/
for(gt=numberOfGTs; gt>0; gt-=auxPopSize) {
  numberOfNewBacteria = gt>auxPopSize?auxPopSize:gt;
  for(auxIndex=0; auxIndex<numberOfNewBacteria; auxIndex++) {
    src = rnd(0, popSize);  /* rnd(a, b) returns an integer in the range [a; b) */
    dest = rnd(0, popSize);
    /* The bacteria with smaller objective fn. value is better */
    if(objectiveValue(population[src]) > objectiveValue(population[dest])) {
      commute(src, dest);
    }
    newBact = population[dest];
    gene = rnd(0, numberOfGenes);
    newBact[gene] = src[gene];  /* transferring a gene from source to destination */
    auxPop[auxIndex] = newBact;
  }
  evaluateObjectiveFnsParallel(auxPop);
  sort(auxPop);
```

```
    /* copy the best popSize pieces of bacteria from population and auxPop to population
*/
    merge(population, auxPop, popSize);
}
```

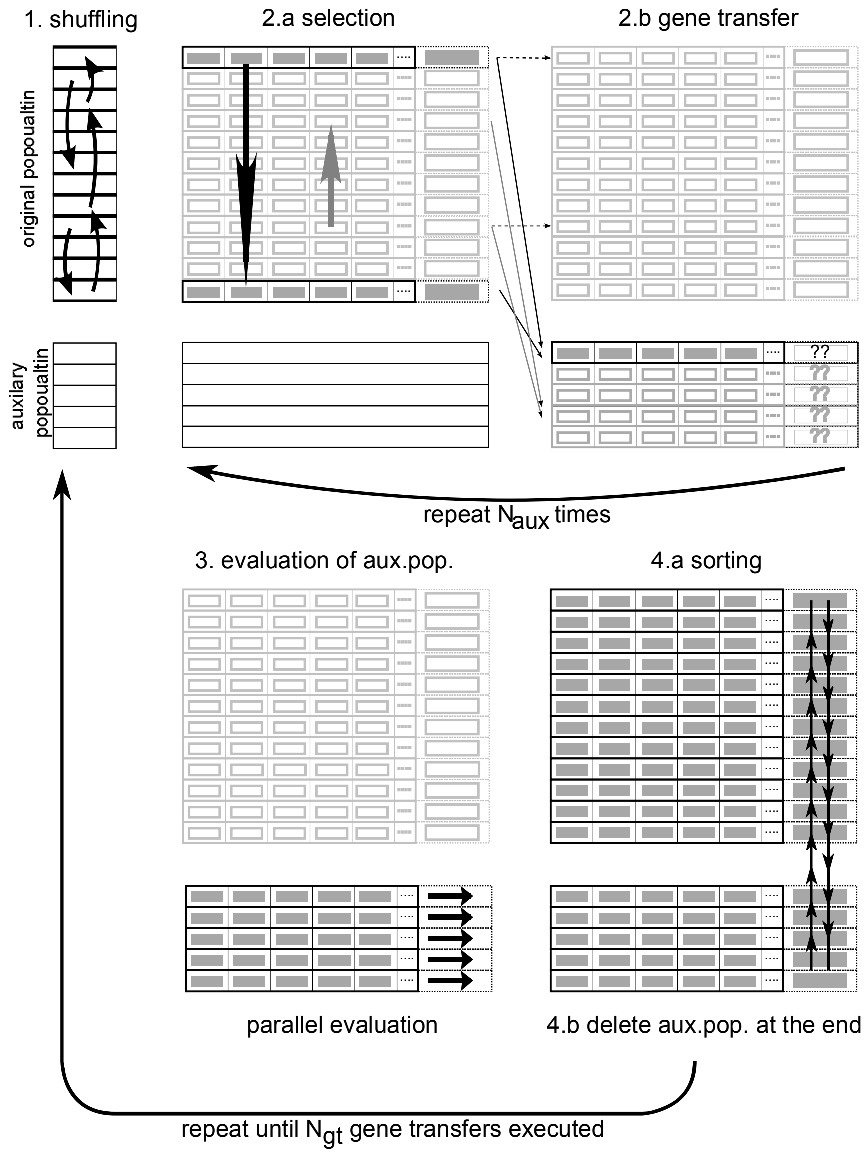

Figure 6. Operation of the gene transfer inspired by MGA with auxiliary population

All the gene transfers above can be improved with a simple modification. Copying more genes from the source bacteria to the destination at once typically increases the speed of convergence, but the chance of premature convergence as well. The analysis of this phenomenon is not subject of this article, but we remark that we used identical settings throughout the execution of the test functions to avoid the side effects.

3. Test functions

The working of the original and the suggested three new gene transfer operators was tested on artificial test functions. These well-known functions [3][12][16] were selected on the basis of their properties. Table 1 summarizes some properties of the functions.

Table 1. Properties of the test functions

	Linearity	**Modality**	**Continuity**	**Plateaux/shoulder**
De Jong's 1st (sphere)	yes	unimodal	yes	no
De Jong's 3rd	yes	multimodal	no	yes
Step	yes	multimodal	no	yes
Rastrigin's	no	multimodal	yes	no
Keane's	no	multimodal	no	no

The equations of the test functions are the following.

$$f_{DeJong1} = \sum_{i=1}^{n} x_i^2, -5.12 \le x_i \le 5.12 \tag{1}$$

$$f_{DeJong3} = \sum_{i=1}^{n} \lfloor x_i \rfloor, -5.12 \le x_i \le 5.12 \tag{2}$$

$$f_{Step} = \sum_{i=1}^{n} \lfloor x_i + 0.5 \rfloor^2, -5.12 \le x_i \le 5.12 \tag{3}$$

$$f_{Rastrigin} = n \cdot 10 + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) \right), \quad -5.12 \le x_i \le 5.12 \tag{4}$$

$$f_{Keane} = \left| \frac{\left(\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i) \right)}{\sqrt{\sum_{i=1}^{n} i x_i}} \right|, \quad \begin{array}{l} \prod_{i=1}^{n} x_i \ge 0.75, \\ \sum_{i=1}^{n} x_i \le 7.5n, \end{array} \quad 0 \le x_i \le 10 \tag{5}$$

4. Test environment of the measurements

The BEA used in the measurements was a custom implementation, which has some interesting properties.

- 20 floating point genes have been used because the aim is to investigate optimization methods of continuous problems arising in engineering.
- The software is able to handle multiple stop conditions, but only the wall clock time limit of the simulation was used. The application terminates only if all the operations of the current population are finished. All the measurements were repeated 20 times, and the average of them is considered a result. The standard deviation of these values was also calculated to determine the statistical uncertainty. Note that the artificial delay in the objective function is necessary, if we want to measure how efficient a BEA-like method in complex problems will be, where the evaluation of individuals take much more time than the communication between computational nodes. However, the standard test functions above are better for testing than complex real-life problems, because their qualitative properties and global maximum is well known from the literature and their CPU-usage is limited, which allows us to perform a high number of test calculations.
- Master-slave style parallelism was used in order to speed up the optimisation process. The master CPU was responsible to execute all the bacterial operators and handle the population. The only job of the slave CPUs were to calculate the values of the objective function of the bacteria. Because the test functions are relative easy to evaluate, a small delay (approx. 0.005 second) was built in the objective function. Without this modification the overhead of the communication between the CPUs would be disproportionately high and could distort our results.
- The communication between the CPUs has been realised with MPICH 1.0, an implementation of the Message Passing Interface.

The software ran on a cluster of workstations containing 1 master + 16 slave computers connected with Ethernet network. Nine of the slave machines were equipped with Intel Pentium IV CPUs running at 2.4 GHz, the other machines were built with more up-to-date Hyper-Threaded Intel Pentium IV CPUs running at 3.0 GHz. The difference in the computing power of the slave machines did not mean a problem, because the delay of the objective function and the small amount of code practically removed all the differences.

5. Test results

5.1. Rastrigin function

Due to the size limits of the paper, it is impossible to display all data in maximum detail. We decided to show most of the results of the test calculations with the Rastrigin function, but from the rest of the functions we give a short summary only.

The global minimum of the Rastrigin function is at $x_i = 0$ for all i values, and here the value of the objective function is 0. (Every other places $f_{Rastrigin}$ has positive value.) Therefore if we show the best objective function value (B) of the population as a function of wall-clock time (T_{wall}), we get curves that asymptotically converge to 0. The faster the curve approaches zero, the better the method is. We can observe this nature in Figure 7, 8 and 9, which shows the results for 1, 4 and 16 processors respectively.

We can conclude that during the optimisation of the Rastrigin function, the original gene transfer performs well if only one CPU is available, and the MGA inspired gene transfer needs the most time to finish. On the other hand, the performance of the original gene transfer is very weak if it is running on more CPUs. All the three new alternatives behave very well, but in general, the ones with auxiliary population end their job faster, however the difference between them is not significant.

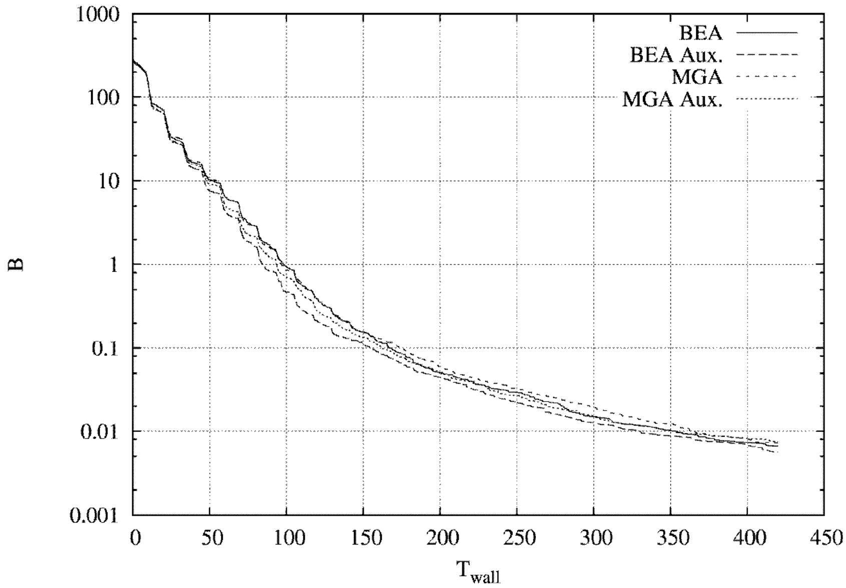

Figure 7. Optimisation of Rastrigin's function, using 1 CPU

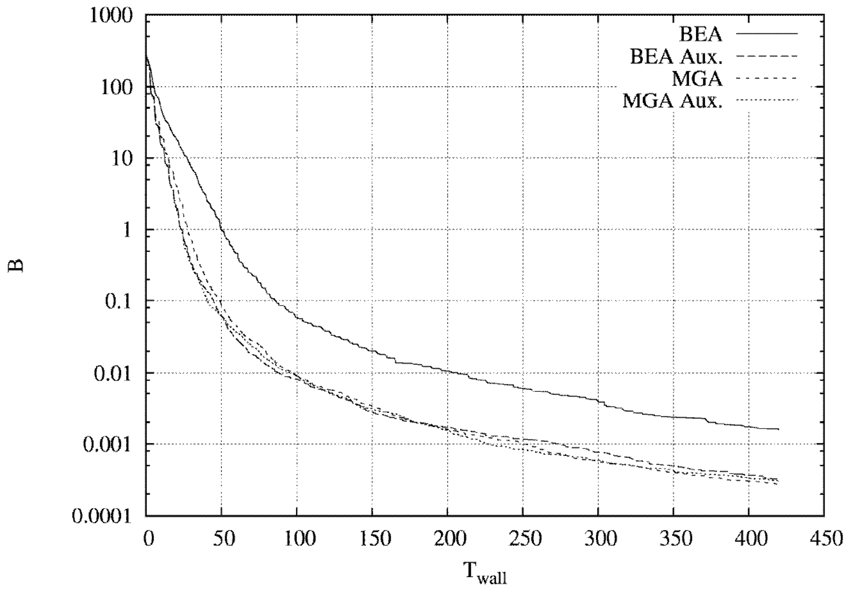

Figure 8. Optimisation of Rastrigin's function, using 4 CPUs

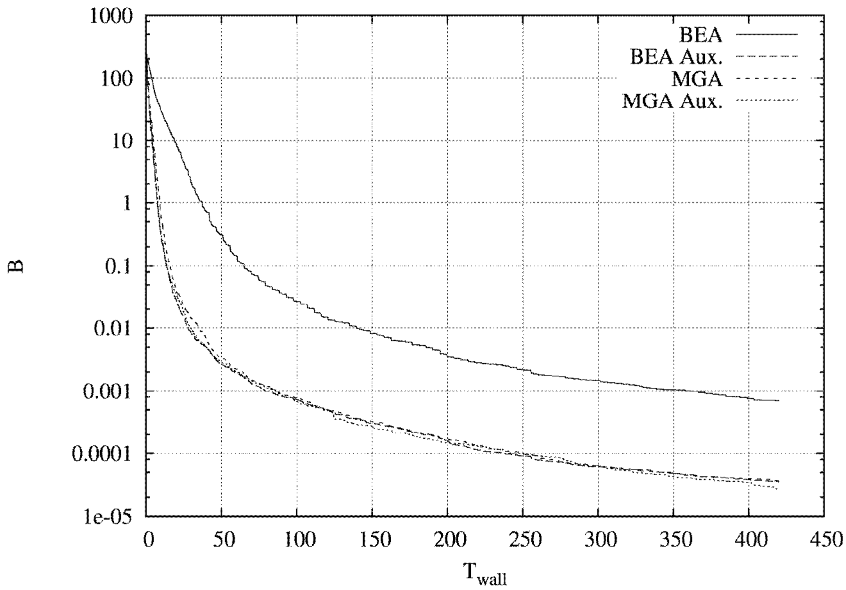

Figure 9. Optimisation of Rastrigin's function, using 16 CPUs

Figure 10 shows how the original BEA scales for $N_{cpu} = 1,2,4,8,16$. However, even if one can observe some improvement at every increase in the number of processors, the scaling is far from ideal. On the other hand, BEA with auxiliary population scales very well, which is demonstrated in Figure 11.

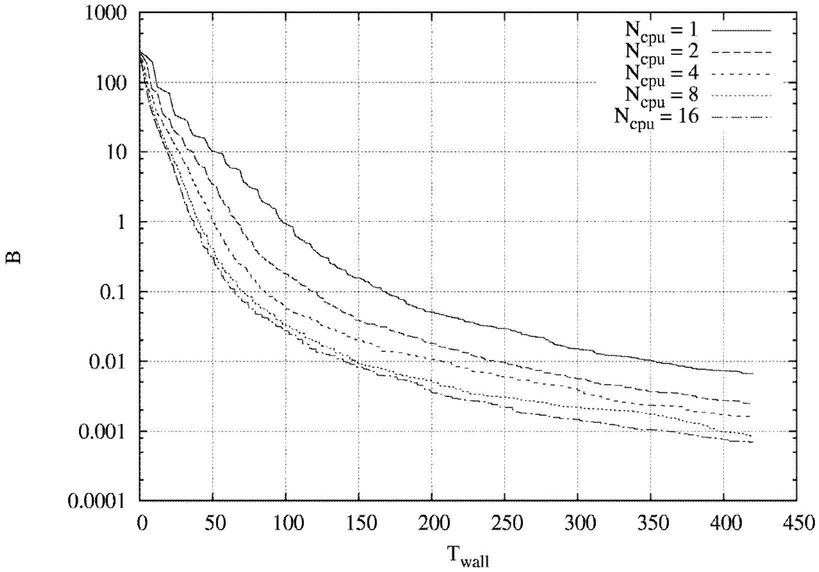

Figure 10. Optimisation of Rastrigin's function, using original gene transfer operation of BEA

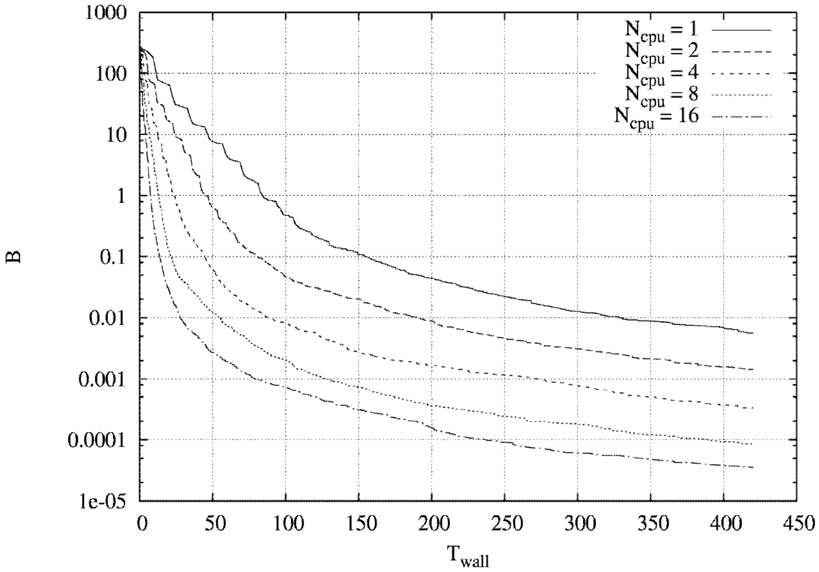

Figure 11. Optimisation of Rastrigin's function, using original gene transfer operation of BEA with auxiliary population

The effectiveness of the proposed methods can be shown in a different way too. We can show the time needed to reach a specific value of the objective function which is near the optimum. We performed calculations for 3 different values (0.05, 0.15 and 0.01). They gave qualitatively the same results, therefore here we only show the T_{wall} needed to reach 0.01. Figure 12 shows this wall-clock time as a number of processors for the 4 examined method.

In an ideal case the measured points in Figure 12 will be on straight lines with slope –1. This is clearly not fulfilled in the original BEA, but the other methods show this nature approximately.

Note that we visualized the uncertainty of measured values in Figure 12. The small error bars show the standard deviation of the average values of the 20 different test calculations.

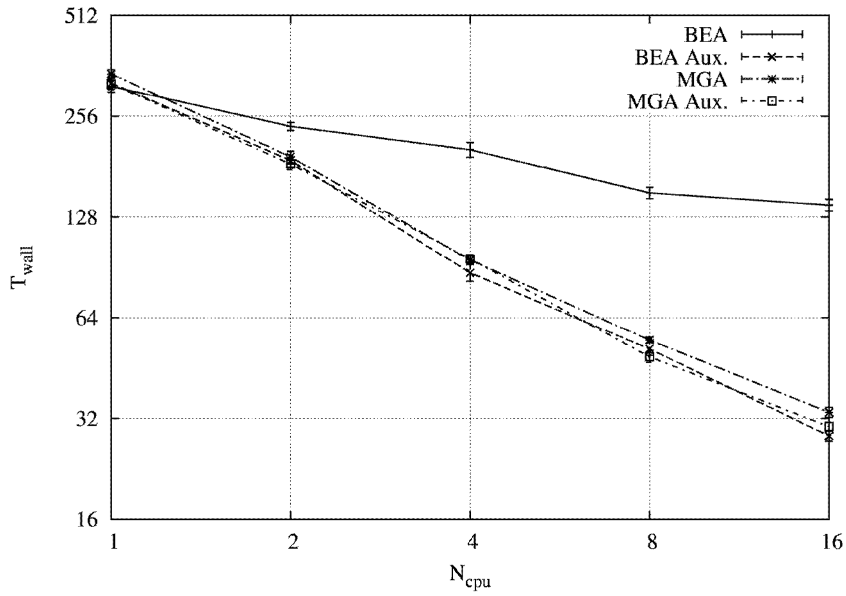

Figure 12. Optimisation of the Rastrigin function with different gene transfer methods and different number of CPUs

It is more precise but less expressive to show the numerical values of test calculations. Table 2 shows the wall-clock time, its standard deviation, the calculated parallel effectiveness and its standard deviation for the same case as in Figure 12. The corresponding tables for 0.015 and 0.05 objective functions value were calculated, but as they are significantly similar and the 0.01 case is so strong, that we do not publish them to keep the extent of paper reasonable.

Table 2. Time and efficiency values of the optimisation of the Rastrigin function. The software stopped when the objective value reached 0.01.

	BEA	BEA Aux.	MGA	MGA Aux.
T*	314.461	321.165	342.870	319.934
σ(T*)	12.814	13.440	10.232	10.748
T_2	238.598	189.014	193.519	184.372
σ(T_2)	6.601	4.699	7.537	7.159
E_2	0.659	0.850	0.886	0.868
σ(E_2)	0.032	0.041	0.043	0.045
T_4	203.384	87.132	95.193	95.989
σ(T_4)	10.219	4.835	2.535	2.825
E_4	0.387	0.921	0.900	0.833
σ(E_4)	0.025	0.064	0.036	0.037
T_8	151.276	51.559	54.813	48.813
σ(T_8)	5.977	2.270	1.082	1.875
E_8	0.260	0.779	0.782	0.819
σ(E_8)	0.015	0.047	0.028	0.042
T_{16}	139.205	28.382	33.388	30.294
σ(T_{16})	5.575	0.998	1.163	0.919
E_{16}	0.141	0.707	0.642	0.660
σ(E_{16})	0.008	0.039	0.029	0.030

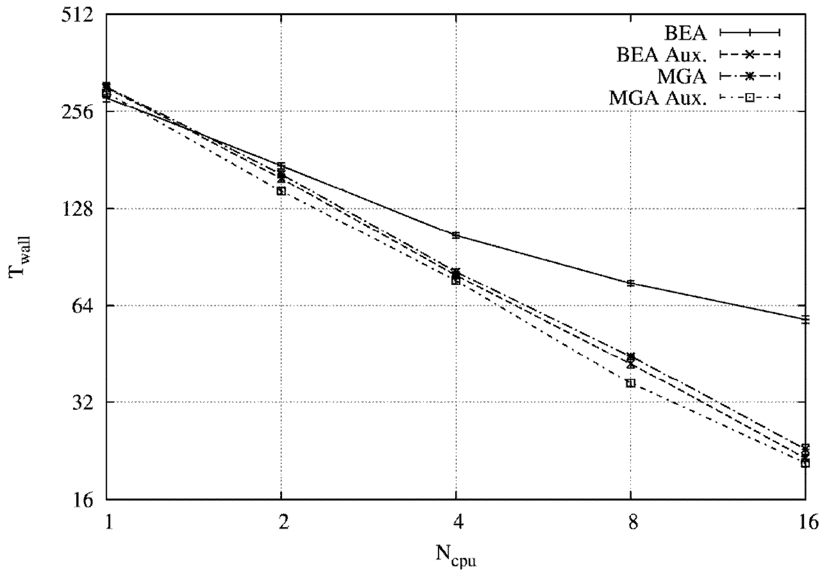

Figure 13. Optimisation of the Keane function with different gene transfer methods and different number of CPUs

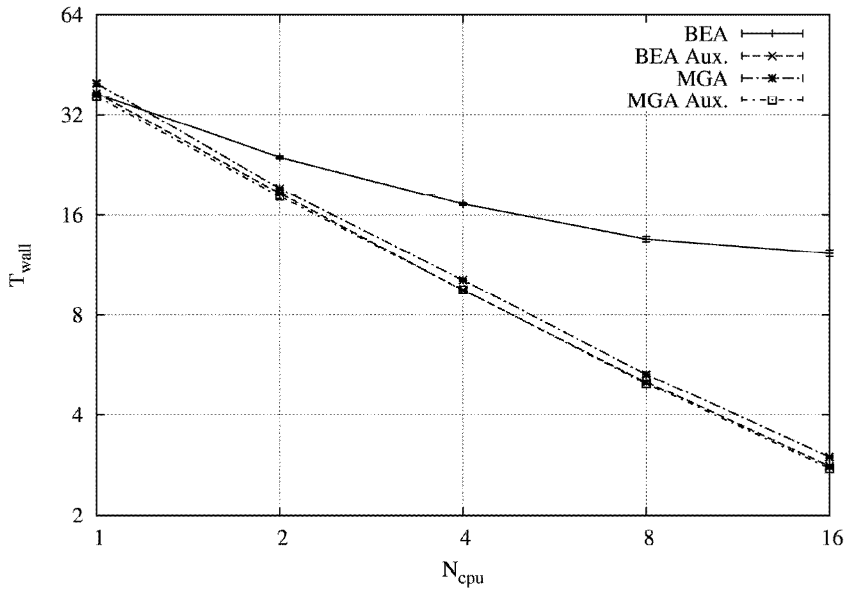

Figure 14. Optimisation of the Step function with different gene transfer methods and different number of CPUs

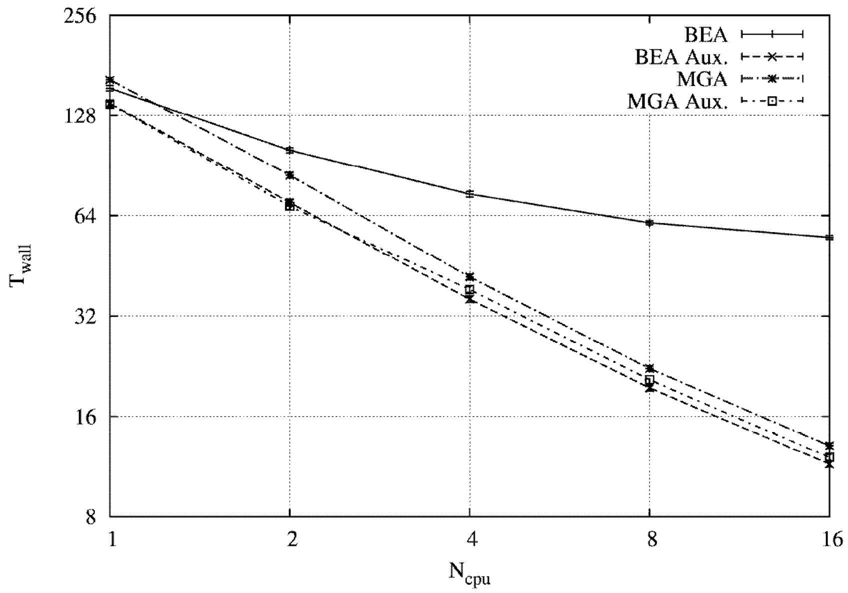

Figure 15. Optimisation of De Jong's 1st function with different gene transfer methods and different number of CPUs

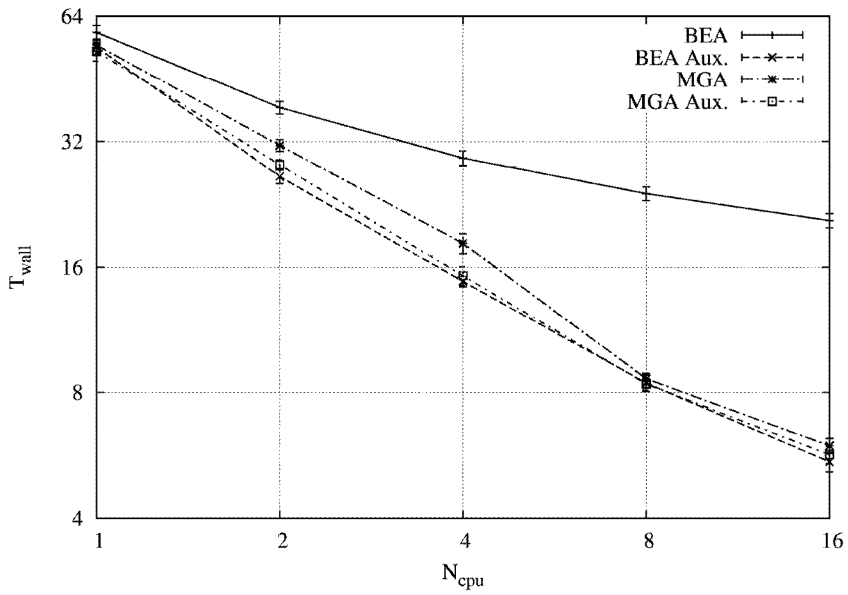

Figure 16. Optimisation of De Jong's 3rd function with different gene transfer methods and different number of CPUs

5.2. Other test functions

Here we present a small number of graphs and tables from the other 4 test functions.

Figure 13, 14, 15 and 16 are the correspondent graphs of Figure 12.

We can draw similar inferences in connection with the optimisation of the Keane function as well. With one CPU, the original gene transfer is competitive, but using more CPUs, its arrears are evident. The winner of the optimisation is the MGA inspired gene transfer with auxiliary population, according to wall clock time and efficiency. The second best method is the original gene transfer with auxiliary population. Note, that the difference between methods is not significant: the error bars of the standard deviation of values are often overlapping.

Table 3. Time and efficiency values of the optimisation of the Keane function. The software stopped when the objective value reached 0.76.

	BEA	BEA Aux.	MGA	MGA Aux.
T*	282.408	304.116	305.870	293.280
T_2	173.776	158.711	163.589	144.986
E_2	0.813	0.958	0.935	1.011
T_4	105.415	79.412	81.359	76.441

	BEA	BEA Aux.	MGA	MGA Aux.
E_4	0.670	0.957	0.940	0.959
T_8	74.974	42.197	44.603	36.697
E_8	0.471	0.901	0.857	0.999
T_{16}	57.980	21.524	22.924	20.803
E_{16}	0.304	0.883	0.834	0.881

Table 3 shows wall-clock time needed to reach the objective function value 0.76 and efficiency values for Keane's test function. For simplicity, we neglected the standard variances of the displayed values. (The relative standard deviations were between 1 and 4% for T_{wall} and 3-5% for efficiency.)

Not surprisingly, the optimisation of the step function has ended with a similar result. The two kinds of gene transfer with auxiliary population performed practically the same, and they are the winner of the optimisation. (The relative standard deviations were 0-2% for T_{wall} and 3-5% for efficiency.) The MGA inspired gene transfer was the slowest of the three new alternative methods, but its high efficiency is remarkable. Its disadvantage decreases by the increase of the number of the CPUs. For much higher number of processors, MGA may be the winner.

Table 4. Time and efficiency values of the optimisation of the step function. The software stopped when the objective value reached 0.

	BEA	BEA Aux.	MGA	MGA Aux.
T*	37.011	37.223	39.888	36.342
T_2	23.816	18.590	19.234	18.220
E_2	0.777	1.001	1.037	0.997
T_4	17.257	9.485	10.184	9.516
E_4	0.536	0.981	0.979	0.955
T_8	13.538	5.032	5.316	4.981
E_8	0.342	0.925	0.938	0.912
T_{16}	12.272	2.808	2.991	2.768
E_{16}	0.188	0.828	0.833	0.820

In case of optimising De Jong's 1st function, it is worth using gene transfers with auxiliary population even in the case of one processor (see Table 5). MGA inspired gene transfer is the slowest of the three new operations, but its high efficiency helps reducing its disadvantages if high number of processor is available. (The relative standard deviations were 1-3% for T_{wall} and 2-3% for efficiency.)

Table 5. Time and efficiency values of the optimisation of De Jong's 1st function. The software stopped when the objective value reached 0.001.

	BEA	BEA Aux.	MGA	MGA Aux.
T*	154.714	139.269	164.104	138.584
T_2	100.573	69.999	84.420	68.255
E_2	0.769	0.995	0.972	1.015
T_4	74.202	35.933	41.883	38.397
E_4	0.521	0.969	0.980	0.902
T_8	60.917	19.485	22.279	20.637
E_8	0.317	0.893	0.921	0.839
T_{16}	55.086	11.497	13.069	12.041
E_{16}	0.176	0.757	0.785	0.719

With optimising De Jong's 3rd function (Table 6), all new operations are faster than the original gene transfer, and the difference grows by the increase of the number the processors. The two operations with auxiliary population are the best in this comparison. (The relative standard deviations were 3-5% for T_{wall} and 4-7% for efficiency.)

Table 6. Time and efficiency values of the optimisation of De Jong's 3rd function. The software stopped when the objective value reached 119.9.

	BEA	BEA Aux.	MGA	MGA Aux.
T*	58.537	53.980	54.566	52.711
T_2	38.584	26.380	31.309	28.152
E_2	0.759	1.023	0.871	0.936
T_4	29.203	14.786	18.261	15.257
E_4	0.501	0.913	0.747	0.864
T_8	24.018	8.422	8.622	8.382
E_8	0.305	0.801	0.791	0.786
T_{16}	20.701	5.449	5.956	5.669
E_{16}	0.177	0.619	0.573	0.581

6. Conclusion

Based on the extended test calculations with 5 different standard test functions, we can conclude that all the three proposed alternative gene transfer operations are well suited for parallel execution. In general, the auxiliary population increased the speed of convergence, thus its use is recommended. The MGA inspired gene transfer performs better with the auxiliary population, but its simplicity and high efficiency makes its implementation a reasonable decision without auxiliary population, too. In many cases, the usage of some of the new alternative methods is effective even if only one CPU is available.

For complex objective functions (e.g. the ones that arise in engineering optimization problems) the original BEA is a bad choice due to its bad scaling properties. For at most 16 processors the BEA or MGA-based method with auxiliary population appears to be the good choice. (There is no significant difference between them.)

The efficiency values show that for some cases the MGA-like version (without auxiliary population) has the best scaling from the four methods examined here, therefore it is possible, that for much higher number of processors, MGA will be the winner in this case. We will examine this effect in a subsequent paper.

Acknowledgement

The authors' research is supported by the National Development Agency and the European Union within the frame of the project TAMOP 4.2.2-08/1-2008-0021 at the Széchenyi István University entitled "Simulation and Optimization – basic research in numerical mathematics".

References

[1] Bäck, T., Fogel, D. B., Michalewicz, Z.: *Handbook of Evolutionary Computation*, IOP Publishing and Oxford University Press, (1997).

[2] Botzheim, J., Cabrita, C., Kóczy, L. T., Ruano, A. E.: *Fuzzy rule extraction by bacterial memetic algorithms*, Proceedings of the 11th World Congress of International Fuzzy Systems Association, IFSA 2005, Beijing, China, (July 2005), pp. 1563–1568.

[3] De Jong, K. A.: *An analysis of the Behavior of a Class of Genetic Adaptive Systems*, dissertation, University of Michigan, (1975).

[4] Farkas, M., Földesi, P., Botzheim, J., Kóczy, L. T.: *A Comparative Analysis of Different Infection Strategies of Bacterial Memetic Algorithms*, INES 2010 14th International Conference on Intelligent Engineering Systems, Las Palmas of Gran Canaria, Spain, (May 5–7, 2010).

[5] Goldberg, D. E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Inc., (1989).

[6] Grosan, C., Abraham, A.: *Hybrid Evolutionary Algorithms: Methodologies, Architectures, and Reviews*, Studies in Computational Intelligence, Vol. 75/2007, (2007), pp. 1-17.

[7] Harvey, I.: *The Microbial Genetic Algorithm*, in Proceedings of the Tenth European Conference on Artificial Life, editor G. Kampis et al, Springer LNCS., (2009).

[8] Harvey, I.: *The Microbial Genetic Algorithm*, School of Cognitive and Computing Sciences, University of Sussex, submitted as a Letter to Evolutionary Computation, (1996), unpublished report (http://www.informatics.sussex.ac.uk/users/inmanh/Microbial.pdf)

[9] Horváth, A., Horváth Z.: *Optimal shape design of diesel intake ports with evolutionary algorithm*, Proceedings of 5th European conference on numerical mathematics and advanced applications (ENUMATH 2003), edited by Feistauer, M. et al., Springer Verlag, (2004), pp. 459–470.
[10] Li, K., Pan, Y., Shen, H., Zheng, S. Q.: *A Study of Average-Case Speedup and Scalability of Parallel Computations on Static Networks*, Mathematical and Computer Modelling 29, (1999), pp. 83-94.
[11] Moscato P.: *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*, Caltech Concurrent Computation Program, Tech. Rep., California, (1989).
[12] Mühlenbein, H., Schomisch, D., Born, J.: *The Parallel Genetic Algorithm as Function Optimizer*, Parallel Computing, Vol. 17, (1991), pp. 619–632.
[13] Nawa, N. E., Furuhashi, T.: *Fuzzy System Parameters Discovery by Bacterial Evolutionary Algorithm*, IEEE Transactions on Fuzzy Systems, Vol. 7, No. 5, (1999), pp. 608-616.
[14] Nawa, N. E., Hashiyama, T., Furuhashi, T., Uchikawa, Y.: *A study on fuzzy rules discovery using pseudo-bacterial genetic algorithm with adaptive operator*, Proceedings of IEEE Int. Conf. on Evolutionary Computation, ICEC'97, (1997).
[15] Nawa, N. E., Furuhashi, T.: *A Study on the Effect of Transfer of Genes for the Bacterial Evolutionary Algorithm*, Second Intenational Conference on Knowledge-Based Intelligent Electronic System, editors Jain, L. C., Jain, R. K., Adelaide, Australia, (21-23 April 1998), pp. 585-590.
[16] Törn, A., Zilinskas, A.: *Global Optimization*, Lecture Notes in Computer Science, No. 350, Springer-Verlag, Berlin, (1989).