BIOINSPIRED OPTIMIZATION METHODS AND THEIR APPLICATIONS

Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Applications, BIOMA 2006

9–10 October 2006, Ljubljana, Slovenia

Edited by
BOGDAN FILIPIČ
JURIJ ŠILC

Jožef Stefan Institute, Ljubljana

Jožef Stefan Institute
Ljubljana, Slovenia

Preface

Faced with insufficient performance of traditional computational methods on demanding real-world problems, computer scientists have decades ago started designing a novel class of problem solving techniques inspired by biological phenomena, such as collaboration and competition among individuals in a struggle for limited resources, recombination and propagation of genetic material from generation to generation, and emergent behavior of insect colonies and bird flocks. Simplified models of these mechanisms are nowadays employed in problem solving techniques, known as evolutionary computation, ant colony optimization, particle swarm optimization and others, that alleviate the shortcomings of traditional algorithms in large-scale applications where little is known about the properties of the underlying problems. Moreover, the bioinspired techniques are becoming increasingly popular for their robustness, capability of providing alternative solutions and amenability to implementation in distributed computing environments. It is therefore not surprising that they are being regularly used in tackling search and optimization tasks in science, engineering and business.

This volume contains some of the recent theoretical and practical contributions to the field of bioinspired optimization. The papers were presented at the Second International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2006), held in Ljubljana, Slovenia, on 9 and 10 October 2006. Encouraged by the success of the first BIOMA in 2004, we organized the conference again to bring together theoreticians and practitioners to present their recent achievements in a single stream of talks, and exchange the ideas in informal discussions. After the review process, 16 papers were accepted for publication, contributed by 35 (co)authors coming from 7 countries.

Professor Günter Rudolph from the University of Dortmund, widely known for his numerous theoretical studies of evolutionary algorithm properties, delivered an invited talk on deployment scenarios of parallelized code in stochastic optimization. The remaining contributions were divided into two categories, one dealing with theoretic and algorithmic issues, and the other presenting practical applications. Theoretical and algorithmic studies address specialized topics in bioinspired optimization: entropy driven exploration and exploitation in evolutionary algorithms, niching for multimodal optimization problems, self-adaptation in differential evolution, stopping criteria for constrained optimization with particle swarms, a non-parametric genetic algorithm, takeover time in evolutionary algorithms processing parallel subpopulations with migrating individuals, and stigmergy as a numerical optimization concept. Reports on applied work come from a variety of application domains: dietary menu planning, optimal mission planning for autonomous unmanned aerial vehicles, database index optimization, job scheduling on computational grids, optimization of metallurgical production processes, characterization of biosystem complexity with electron paramagnetic resonance, and genetic programming of sensor networks.

BIOMA 2006 was sponsored by the Slovenian Research Agency. It was organized as part of the 9th International Multiconference Information Society (IS 2006) taking place at the Jožef Stefan Institute, Ljubjana, from 9 to 14 October 2006. BIOMA was held at the Jožef Stefan International Postgraduate School that also includes bioinspired optimization in its curriculum.

We are grateful to the conference sponsors, members of the program and organizing committees, the invited speaker, and regular paper presenters for taking part in shaping the conference. We hope you find the event enjoyable and the book inspiring, and invite you to meet again at the next BIOMA.

Ljubljana, 25 September 2006

BOGDAN FILIPIČ AND JURIJ ŠILC

DEPLOYMENT SCENARIOS OF PARALLELIZED CODE IN STOCHASTIC OPTIMIZATION

Günter Rudolph
Department of Computer Science
University of Dortmund, Germany
guenter.rudolph@uni-dortmund.de

Abstract The benefit of using parallel hardware in real-time environments is obvious: For example, if it is necessary to solve some optimization task periodically in a narrow time window a parallelized optimization algorithm can possibly meet the time constraints. In case of deterministic algorithms the situation is clear. But if we use randomized algorithms some questions appear: As randomized algorithms must be run more than once to get a reliable solution we can execute the sequential code in parallel independently or we can execute the parallelized code simultaneously on the parallel hardware in a successive manner. Which approach is better? We analyze several scenarios analytically and offer conditions for deciding when to deploy the parallelized code and when not.

Keywords: Parallel optimization, Randomized algorithms, Stochastic optimization

1. Introduction

The utility of a parallelized deterministic optimization algorithm is evident: Since the deterministic algorithm is run only once, the parallel version delivers the solution more rapidly. In case of randomized optimization algorithms the situation changes. Typically, these randomized algorithms (RAs) must be run several times to avoid bad results produced by some unlucky sequence of random variables used in the RA. This observation raises the question if the burden of developing a parallel randomized algorithm is worth the effort: Instead of running a parallelized RA several times in sequence on the parallel hardware, one can also run the original sequential code independently in parallel on several processors. Which are the situations in which running the parallelized code is advantageous? And when the recommendation should be the other way round?

Here, we analyze some situations based on certain scenarios. Our main assumption is that we have a periodically appearing optimization task. Therefore it is reasonable to use the *expectation of random variables* for comparisons: If the expected runtime of successive runs of the parallelized code is less than the expected runtime of parallel runs of the sequential code, then and only then it is advisable to deploy the parallelized RA.

This approach also has the appealing aspect that we can elude from the ongoing discussion how to measure the performance of parallelized RAs [1, 2] in terms of speedup, efficiency and related measures.

Here we extend and generalize our findings presented in [4]. For this purpose, Section 2 presents some mathematical results used in the sequel. Sections 3 and 4 present several scenarios and offer conditions for deciding when to deploy the parallelized code and when not. Finally, our conclusions can be found in Section 5.

2. Mathematical Preliminaries

Let $X_1, X_2, \ldots, X_p$ be independent and identically distributed (i.i.d.) random variables. Their minimum and maximum are denoted by $X_{1:p} = \min\{X_1, X_2, \ldots, X_p\}$ and $X_{p:p} = \max\{X_1, X_2, \ldots, X_p\}$, respectively. For certain distributions of the X_k the expectation of the minimum and maximum can be calculated analytically. For example [3, p. 35], if the X_k are uniformly distributed in the interval $[\,a, b\,]$ then

$$\mathsf{E}[\,X_k\,] = \frac{b-a}{2}, \quad \mathsf{V}[\,X_k\,] = \frac{(b-a)^2}{12}\,,$$

$$\mathsf{E}[\,X_{1:p}\,] = a + (b-a)\,\frac{1}{p+1} \quad \text{and} \quad \mathsf{E}[\,X_{p:p}\,] = a + (b-a)\,\frac{p}{p+1}\,. \tag{1}$$

Moreover, there exist numerous inequalities for the expectations, each of them based on some assumptions. The most general inequality is probably given in [3, p. 59 and 63] since it only assumes the existence of the second moment.

Theorem 1
Let $X, X_1, X_2, \ldots, X_p$ be i.i.d. random variables with $\mathsf{E}[\,X^2\,] < \infty$. Then

$$\mathsf{E}[\,X\,] - \frac{p-1}{\sqrt{2\,p-1}}\,\mathsf{D}[\,X\,] \le \mathsf{E}[\,X_{1:p}\,] \le \mathsf{E}[\,X_{p:p}\,] \le \mathsf{E}[\,X\,] + \frac{p-1}{\sqrt{2\,p-1}}\,\mathsf{D}[\,X\,]$$

where $\mathsf{D}[\,X\,]$ denotes the standard deviation of X. $\square$

Another result that will be useful is known as Wald's equation. A proof can be found e.g. in [5, p. 166f].

Theorem 2
Let N be a positive, integer-valued random variable and $X_1, X_2, \ldots$ be an i.i.d. sequence of random variables where N is also independent of the X_k. Then the expectation and variance of the random sum consisting of the first N members

of the X_k are given by

$$\mathsf{E}\left[\sum_{k=1}^{N} X_k\right] = \mathsf{E}[\,N\,]\cdot\mathsf{E}[\,X_1\,] \tag{2}$$

$$\mathsf{V}\left[\sum_{k=1}^{N} X_k\right] = \mathsf{E}[\,N\,]\cdot\mathsf{V}[\,X_1\,] + \mathsf{V}[\,N\,]\cdot\mathsf{E}[\,X_1\,]^2 \tag{3}$$

where $\mathsf{V}[\,\cdot\,]$ denotes the variance. □

3. Scenario: Run RA Multiple Times, Choose Best Solution Found

In practice, nobody runs a randomized algorithm only once. Rather, the RA is run multiple times and the best solution found within some time limit is used. Figure 1 illustrates our two options how to use the parallel hardware.

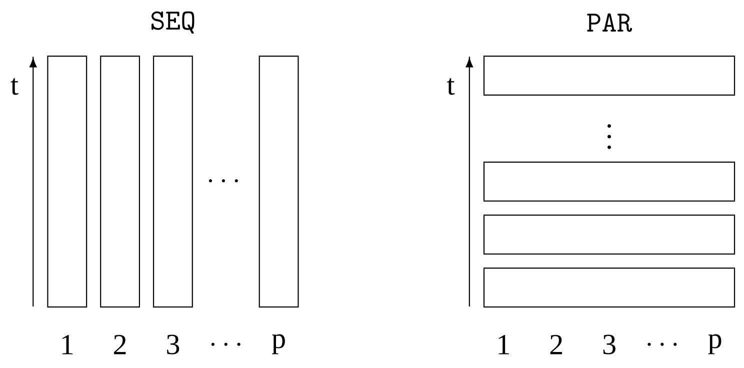


Figure 1. Left: The sequential code is run independently in parallel on p processors. Right: The parallelized code is run on p processors simultaneously for p successive runs.

3.1 Fixed Iteration Number

Let t be the running time of the sequential algorithm and $t_p = c\,t/p$ the running time of the parallelized sequential algorithm, where $c > 1$ aggregates the communication and other overhead costs of the parallelized version. Let n be the maximum number of times we can run the RA before we must use the best solution found and assume that $n = p$ where p is the number of processors.

Then $r = t$ is the total running time of running the sequential algorithm on p processors in parallel. Since the total running time of p successive runs of the parallelized version is $r_p = p \times t_p = c\,t$ we can see easily that nothing is gained by a parallelization. Even worse, every effort invested in this task is a waste of resources.

3.2 Random Iteration Number

The situation changes if the running time of the RA is a random variable. For instance, this may be caused by some stopping rule that is independent from the iteration counter. Let T be the random running time of the sequential algorithm and $T_p = c\,T/p$ the running time of the parallelized sequential algorithm with $c > 1$. Again, assume $n = p$. Then the random total running time R of running the sequential algorithm on p processors in parallel is

$$R = \max\{\, T(1), T(2), \ldots, T(p)\,\} = T_{p:p}$$

where $T(i)$ is the running time at processor i. Clearly, the $T(i)$ are independent and identically distributed. Assume that $T(i)$ is normally distributed with mean $t > 0$ and variance σ^2. Then the expectation of R can be approximated [3] via

$$\mathsf{E}[\,R\,] = \mathsf{E}[\,T_{p:p}\,] \approx \mathsf{E}[\,T\,] + \mathsf{D}[\,T\,]\,\sqrt{2\,\log\,p}\,. \tag{4}$$

The random total running time R_p of p sucessive runs of the parallelized version is given by

$$R_p = \sum_{i=1}^{p} T_p(i) = \frac{c}{p}\sum_{i=1}^{p} T(i)$$

with expectation

$$\mathsf{E}[\,R_p\,] = c\,\mathsf{E}[\,T\,]\,.$$

Thus, the parallelized version is faster if

$$\mathsf{E}[\,R_p\,] < \mathsf{E}[\,R\,] \Leftrightarrow c < 1 + \frac{\mathsf{D}[\,T\,]}{\mathsf{E}[\,T\,]} \;\times\; \sqrt{2\,\log\,p}\,. \tag{5}$$

In other words, the larger is the coefficient of variation $\nu = \mathsf{D}[\,T\,]/\mathsf{E}[\,T\,]$ the larger the benefit achieved by the parallelization of the sequential algorithm! As seen from this analysis, this scenario can be an appropriate field of deployment of parallelized RAs.

One may object that the conclusions drawn from the relationship in Eqn. (5) are shaky since Eqn. (4) is an approximation only. In order to invalidate this objection we first consider an example for which the result can be reproduced exactly in analytical manner. Next we generalize the result by means of Theorem 1.

Assume that $T(i) \sim U(t-\varepsilon, t+\varepsilon)$ are uniformly distributed in the interval $[\,t-\varepsilon, t+\varepsilon\,]$ for some $t, \varepsilon > 0$. For sake of brevity we shall write T instead of

$T(i)$. Insertion in Eqn. (1) yields

$$\mathsf{E}[\,T\,] = t\,, \quad \mathsf{V}[\,T\,] = \frac{\varepsilon^2}{3}\,, \quad \mathsf{E}[\,T_{p:p}\,] = t + \varepsilon\,\frac{p-1}{p+1}\,.$$

Thus, $\mathsf{E}[\,R_p\,] < \mathsf{E}[\,R\,]$ if and only if $c\,t \le t + \varepsilon\,(p-1)/(p+1)$ or equivalently

$$c < 1 + \frac{\varepsilon}{t\,\sqrt{3}}\,\frac{p-1}{p+1}\,\sqrt{3} \;=\; 1 + \frac{\mathsf{D}[\,T\,]}{\mathsf{E}[\,T\,]} \times \frac{p-1}{p+1}\,\sqrt{3}\,. \tag{6}$$

For example, if we use 9 processors and the running time is uniformly distributed between 40 and 60 seconds then Eqn. (6) yields $c < 1 + 4/25 = 1.16$. As a consequence, the efficiency $1/c$ of the parallelization must be larger than $25/29 \approx 86.2\,\%$. Otherwise, one should run the sequential code in parallel independently.

Next, we generalize our findings. Comparison of Eqn. (5) and Eqn. (6) reveals the same pattern:

$$c < 1 + \frac{\mathsf{D}[\,T\,]}{\mathsf{E}[\,T\,]} \times g(p) \tag{7}$$

for some function $g(\cdot)$ depending on the number of processors p. In order to derive condition Eqn. (7) analytically recall that the condition originally reads

$$\mathsf{E}[\,R_p\,] < \mathsf{E}[\,R\,] \quad \Leftrightarrow \quad c\,\mathsf{E}[\,T\,] < \mathsf{E}[\,T_{p:p}\,] \quad \Leftrightarrow \quad c < \frac{\mathsf{E}[\,T_{p:p}\,]}{\mathsf{E}[\,T\,]}\,.$$

Evidently, this condition is fulfilled if we bound $\mathsf{E}[\,T_{p:p}\,]$ from above via Theorem 1, that is valid for arbitrary runtime distributions. We obtain

$$c < \frac{\mathsf{E}[\,T_{p:p}\,]}{\mathsf{E}[\,T\,]} \le \frac{\mathsf{E}[\,T\,] + \mathsf{D}[\,T\,] \times \frac{p-1}{\sqrt{2\,p-1}}}{\mathsf{E}[\,T\,]} = 1 + \frac{\mathsf{D}[\,T\,]}{\mathsf{E}[\,T\,]} \times \frac{p-1}{\sqrt{2\,p-1}}$$

confirming that the pattern in Eqn. (7) did not appear by chance. Moreover, we have shown that

$$g(p) \le \frac{p-1}{\sqrt{2\,p-1}}$$

regardless of the runtime distribution of T.

4. Scenario: Run Until Satisfactory Solution Found

One might argue that the previous scenario is not always the case. For example, if we need only a satisfactory solution then we can stop the RA as soon as such a solution has been detected. In principle, this can happen in a single run of the RA. Figure 2 illustrates our two options how to use the parallel hardware.

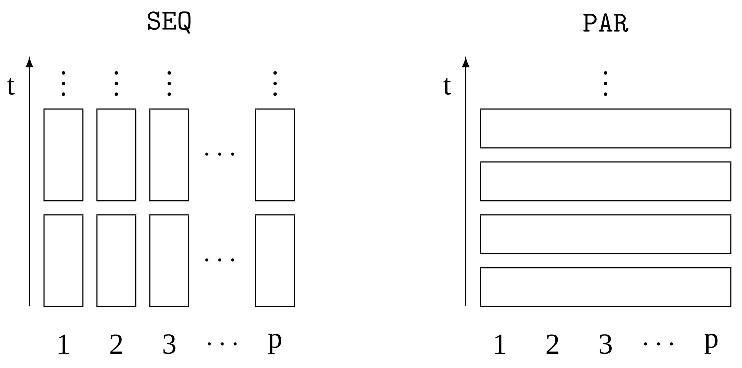


Figure 2. Left: The sequential code is run independently in parallel on p processors until a satisfactory solution is found. Right: The parallelized code is run repeatedly on p processors simultaneously until a satisfactory solution is found.

4.1 Fixed Iteration Number

As in the previous scenario let t be the running time of the sequential algorithm and $t_p = c\,t/p$ the running time of the parallelized sequential algorithm with $c > 1$. Suppose there exists a success probability $s \in (0,1)$ for each run of the RA such that the random variable G represents the number of runs until a successful run occurs. The random variable G has geometrical distribution with probability function

$$\mathsf{P}\{\, G = k \,\} = s\,(1-s)^{k-1}$$

for $k = 1, 2, \ldots$ and $s \in (0,1)$ with

$$\mathsf{E}[\, G \,] = \frac{1}{s} \quad \text{and} \quad \mathsf{V}[\, G \,] = \frac{1-s}{s^2}\,.$$

The time until a successful run occurs on a single processor is $S = t\,G$. Therefore, the random total running time R of running the sequential algorithm on p processors in parallel is

$$R = \min\{\, S(1), S(2), \ldots, S(p) \,\} = S_{1:p} = t\,G_{1:p}$$

where $G_{1:p}$ denotes the minimum of p independent and identically distributed geometrical random variables. According to [6] we have

$$\mathsf{E}[\, G_{1:p} \,] = \frac{1}{1-(1-s)^p} \quad \text{and} \quad \mathsf{V}[\, G_{1:n} \,] = \frac{(1-s)^n}{[\,1-(1-s)^n\,]^2}$$

such that

$$\mathsf{E}[\, R \,] = t\,\mathsf{E}[\, G_{1:p} \,] = \frac{t}{1-(1-s)^p}\,.$$

The random total running time R_p of p successive runs of the parallelized version is given by

$$R_p = t_p\, S = \frac{c}{p}\, t\, S$$

with expectation

$$\mathsf{E}[\, R_p \,] = \frac{c}{p}\, t\, \mathsf{E}[\, S \,] = \frac{c\, t}{s\, p}\,.$$

Since

$$\mathsf{E}[\, R_p \,] < \mathsf{E}[\, R \,] \iff c < \frac{s\, p}{1-(1-s)^p}$$

there are constellations in which a parallelized version is useful. Figure 3 is intended to provide an impression about the interrelationships. For small success probabilities s as one usually faces in optimizations task in which RAs are used as last remedy, the efficiency of the parallel implementation must be extremely high for recommending the deployment of the parallelized code. Especially in real-time environments assumed here it is unlikely to achieve such a high efficiency.

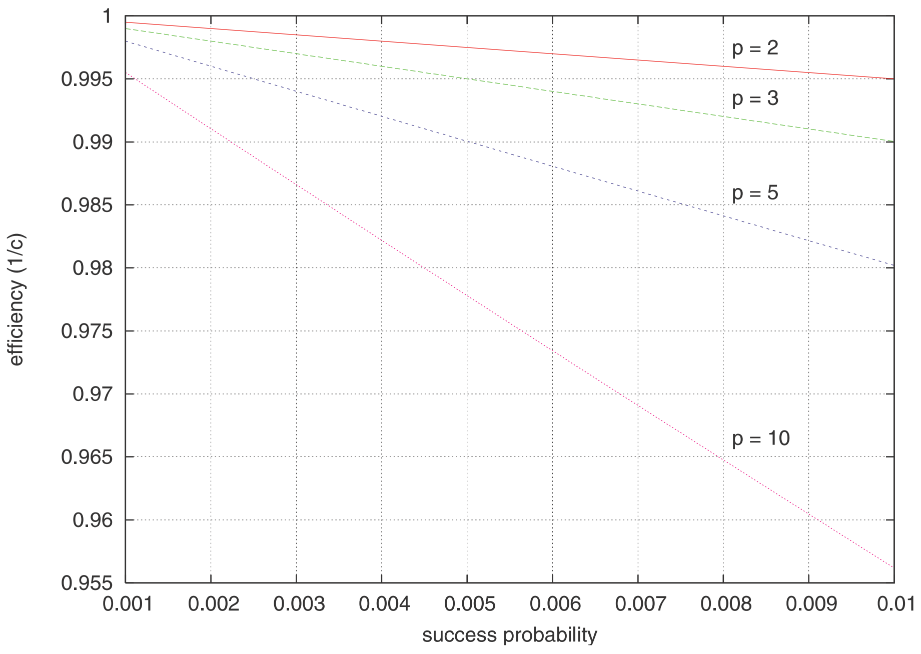


Figure 3. Success probability s versus efficiency $1/c$ of the parallel implementation for some processor numbers.

4.2 Random Iteration Number

Let $T(i)$ be the random running time of run i. Then

$$S = \sum_{i=1}^{G} T(i)$$

is the random time until the first successful run on a single processor. According to Theorem 2 we have $\mathsf{E}[\,S\,] = \mathsf{E}[\,G\,]\,\mathsf{E}[\,T\,]$. As a consequence, the random total running time R of running the sequential algorithm on p processors in parallel is

$$R = \min\{\,S(1), S(2), \ldots, S(p)\,\} = S_{1:p}$$

with

$$\mathsf{E}[\,R\,] = \mathsf{E}[\,S_{1:p}\,] < \mathsf{E}[\,S\,] = \mathsf{E}[\,T\,]\,\mathsf{E}[\,G\,]\,.$$

The random total running time R_p of p successive runs of the parallelized version is given by

$$R_p = \sum_{i=1}^{G} T_p(i) = \frac{c}{p} \sum_{i=1}^{G} T(i)$$

with

$$\mathsf{E}[\,R_p\,] = \frac{c}{p}\,\mathsf{E}[\,T\,]\,\mathsf{E}[\,G\,] = \frac{c}{p}\,\mathsf{E}[\,S\,] = \frac{c\,t}{s\,p}\,.$$

Our condition reads

$$\mathsf{E}[\,R_p\,] < \mathsf{E}[\,R\,] \quad\Leftrightarrow\quad \frac{c}{p}\,\mathsf{E}[\,S\,] < \mathsf{E}[\,S_{1:p}\,]\,.$$

We can express $\mathsf{E}[\,S\,]$ in terms of $\mathsf{E}[\,T\,]$ and $\mathsf{E}[\,G\,]$ but there is a problem for $\mathsf{E}[\,S_{1:p}\,]$: Although we can use the lower bound of Theorem 1 to claim that there is a nonnegative-valued function $h(\cdot)$ with $\mathsf{E}[\,S_{1:p}\,] = \mathsf{E}[\,S\,] - \mathsf{D}[\,S\,] \times h(p)$ and we can express $\mathsf{D}[\,S\,]$ in terms of moments of T and G via Theorem 2, the resulting formula

$$\frac{c}{p}\,\mathsf{E}[\,S\,] < \mathsf{E}[\,S\,] - \mathsf{D}[\,S\,] \times h(p) \quad\Leftrightarrow\quad c < p\left(1 - \frac{\mathsf{D}[\,S\,]}{\mathsf{E}[\,S\,]} \times h(p)\right)$$

does not yield much insight for analyzing the situation.

Therefore we take a look at our condition $\frac{c}{p}\,\mathsf{E}[\,S\,] < \mathsf{E}[\,S_{1:p}\,]$ again. If each T_i has a minimum runtime $a > 0$ then $\mathsf{E}[\,S\,] \geq a\,\mathsf{E}[\,G\,]$ and $\mathsf{E}[\,S_{1:p}\,] \geq a\,\mathsf{E}[\,G\,]$. Since

$$\frac{c}{p}\,\mathsf{E}[\,S\,] \geq \frac{c}{p}\,a\,\mathsf{E}[\,G\,] \to 0 \text{ as } p \to \infty$$

whereas

$$\mathsf{E}[\,S_{1:p}\,] \geq a\,\mathsf{E}[\,G\,] > 0 \text{ regardless of } p$$

we may conclude that there exists a processor number p_0 such that $\mathsf{E}[\, R_p \,] < \mathsf{E}[\, R \,]$ for all $p > p_0$. Thus, this scenario is well suited for parallelized code if many processors are available.

5. Conclusions

We have shown that the recommendation for a deployment of parallelized code depends on several constraints. If we have a fixed time slot and a constant running time of the algorithm then the deployment of parallelized code is a waste of resources. If we can wait until completion of the randomized algorithm which has a random running time, then we need a moderately efficient parallel implementation and a large variation in the running time to favor the parallelized code. If we are in the situation to repeat the algorithm until it fulfills some criterion, then the condition for deploying parallelized code demands a hardly achievable efficiency of the code in case of *constant* running time. If the running time is random then parallelized code may lead to shorter overall running time if many processors are available. The theory in its current state, however, does not yet provide a condition to quantify the number of processors that must be available. Nevertheless, the theory provides some clues that random running times of the randomized algorithms more often lead to recommendations for deploying parallelized code.

References

[1] J. Aczél and W. Ertel. A new formula for speedup and its characterization. *Acta Informatica*, 34(9):637–652, 1997.

[2] E. Alba and A. Luque. Measuring the performance of parallel metaheuristics. In E. Alba, editor, *Parallel metaheuristics: A New Class of Algorithms*, pages 43–62, Wiley, Hoboken, NJ, 2005.

[3] H. A. David. *Order Statistics*. 2nd edition, Wiley, New York, 1981.

[4] G. Rudolph. Parallel evolution strategies. In E. Alba, editor, *Parallel metaheuristics: A New Class of Algorithms*, pages 155–169, Wiley, Hoboken, NJ, 2005.

[5] K. D. Schmidt. *Versicherungsmathematik*. Springer, Berlin et al., 2002.

[6] D. H. Young. The order statistics of the negative binomial distribution. *Biometrika*, 57(1):181–186, 1970.

ENTROPY-DRIVEN EXPLORATION AND EXPLOITATION IN EVOLUTIONARY ALGORITHMS

Shih-Hsi Liu
Department of Computer and Information Sciences
University of Alabama at Birmingham, USA
liush@cis.uab.edu

Marjan Mernik
Faculty of Electrical Engineering and Computer Science
University of Maribor, Slovenia
marjan.mernik@uni-mb.si

Barrett R. Bryant
Department of Computer and Information Sciences
University of Alabama at Birmingham, USA
bryant@cis.uab.edu

Abstract Every evolutionary algorithm needs to address two important facets: exploration and exploitation of a search space. Evolutionary search must combine exploration of the new regions of the space with exploitation of the potential solutions already identified. The necessity of balancing exploration with exploitation needs to be intelligent. This paper introduces an entropy-driven exploration and exploitation approach for evolutionary algorithms. Entropy represents the amount of disorder of the population, where an increase in entropy represents an increase in diversity. New kinds of entropy to express diversity and to control the entropy-driven approach are discussed.

Keywords: Entropy, Evolutionary algorithms, Exploitation, Exploration, Parameter control

1. Introduction

Evolutionary algorithms (EAs) [1, 9] are general purpose searching methods with good yet implicit balance between exploration and exploitation. Exploration is a process of visiting entirely new regions of a search space and of seeing if anything promising may be found in the regions. Exploitation is a process of using information gathered from the previously visited points in the search space to determine which regions might be profitable to be visited next. Exploitation techniques are good at finding local maxima. However, how is the balance between exploration and exploitation achieved in EAs? More importantly, how can the balance be controlled?

In EAs, the selection process and operators (e.g., crossover and mutation) establish a balance between the exploration and exploitation of the search space [5]. A selection process drives searching towards the regions of the best individuals. Hence, exploitation is done by selection. The mutation operator randomly modifies individuals, with a given probability, and thus increases the structural diversity of the population. From this point of view, the mutation operator is more an exploration operator. Such an operator helps to recover the genetic diversity lost during the selection phase and to explore new solutions avoiding premature convergence. On the other hand, mutation can also be seen as an exploitation operator, because most of the genetic material is preserved. However, note that in some EAs (e.g., evolutionary strategies) mutation has a much bigger exploration role than in genetic algorithms. The crossover operator combines two or more parents to generate better offspring. Such a combination can be derived from the idea that the exchange of information between good individuals will generate even better offspring. From this point of view, the crossover operator is more an exploitation operator. However, a good crossover operator should also generate individuals in the exploration zone. Therefore, good balance between exploration and exploitation in EAs is achieved by selection, good mutation/crossover operators and by determining parameters (p_m, p_c, tournament size), which control mutation/crossover and selection, respectively. There have been a variety of studies on determining the best control parameter values [3, 4]. The main problem is to find a set of control parameters, which optimally balances exploration and exploitation: if crossover and mutation rates are very high, much of the space will be explored, but there is a high probability of losing good solutions and of failing to exploit existing schema. If crossover and mutation rates are low, the search space is not explored. The population diversity is therefore rapidly decreasing and ending up in a premature convergence to a non-optimal solution. Despite that, many researchers believed that EAs are effective because of a good ratio of exploration/exploitation. However, this ratio of EAs is implicitly controlled. In some other search techniques such as reinforcement learning [14], one has explicit control over exploration and exploitation. In EAs, one no longer has explicit and respective control over exploitation and exploration, because it is difficult to delimit exploration from exploitation.

In this paper, an entropy-driven exploration and exploitation approach is presented. The exploration/exploitation of the search space is adapted on-line based on the current status of the evolutionary process. The on-line adaptation mechanism involves a decision process as to whether more exploitation or exploration is needed depending on the current progress of the algorithm and on the current estimated potential of discovering better solutions. This decision process is described by a domain-specific language, PPCEA (Programmable Parameter Control for Evolutionary Algorithms) [8]. Because of space consideration, the paper only presents the experimental results using genetic algorithms. Experimenting the mutation role for balancing between exploration and exploitation in evolutionary strategies is our future work.

The paper is organized as follows. Section 2 describes the related work. In Section 3, entropy is introduced to control exploration and exploitation. Section 4 shows the examples and experimental results. Finally, Section 5 presents the conclusion.

2. Related Work

Optimal balance between exploration and exploitation has been mainly controlled by determining the best control parameter values. There are a variety of studies on this topic [4, 7, 8]. Recommendations on control parameters for a particular set of problems can be found in [3, 11]. In [4], an overview of this problem has been given, where the authors distinguish between parameter tuning and parameter control. Furthermore, methods for parameter control have been classified into deterministic, adaptive, and self-adaptive categories: the deterministic category adjusts parameters by deterministic rules; the adaptive category utilizes the feedback of the evolutionary process to control the direction and magnitude of parameters; and the self-adaptive category encodes parameters into individuals and undergoes mutation and recombination.

One of the earliest researchers that investigated entropy in EAs was Rosca [10], whose experiments showed that populations appeared to be stuck in local optima when entropy did not change or decrease monotonically in successive generations. Rosca used fitness values in a population to define entropy and free energy measure. Our work extends Rosca's in trying to find different ways to compute entropy in EAs. Moreover, using entropy as a measure and programmable parameter control by PPCEA [8], we are able to control exploration and exploitation in an adaptable manner.

Diversity-Guided Evolutionary Algorithm (DGEA) [13] uses a distance-to-average-point measure to alternate between phases of exploration and exploitation. It can be expressed easily as a PPCEA program. Moreover, DGEA does not use entropy as a measure for diversity.

3. Entropy in EAs

Entropy is a concept in information theory, thermodynamics, and statistical mechanics. The basic concept of entropy in information theory has to do with how much randomness there is in a signal or random event. Shannon [12] defines entropy in terms of a discrete random event x, with possible states $1..n$ as:

$$H(x) = \sum_{i}^{n} p_i \log_2(\frac{1}{p_i}) = -\sum_{i}^{n} p_i \log_2 p_i. \qquad (1)$$

Statistical mechanics explains entropy as the amount of uncertainty which remains about a system, after its observable macroscopic properties have been taken into account. For a given set of macroscopic quantities, such as temperature and volume, entropy is a function of the probability that the system is in various quantum states. The more states available to the system with higher probability, the greater the disorder and thus, the greater the entropy. If the system has only one possible state, there is no uncertainty, and the entropy of the system is zero. If the system has n possible states which are equiprobable ($p_i = \frac{1}{n}$), the entropy is the highest:

$$H = -n\frac{1}{n} \log_2(\frac{1}{n}) = \log_2 n. \qquad (2)$$

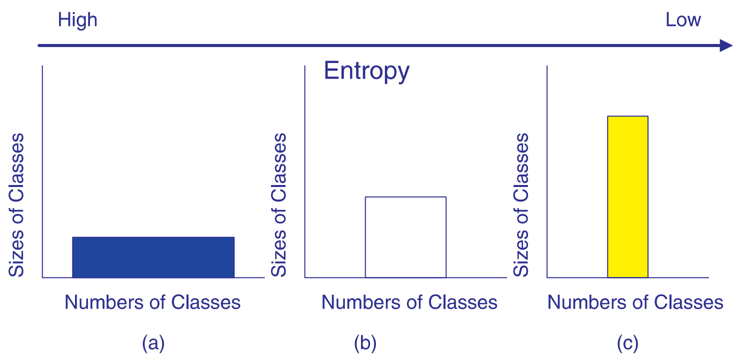


Figure 1. The relationship between entropy and the numbers and sizes of classes.

As such, entropy represents also a succinct measure of diversity. Biological diversity refers to the differences between individuals in a population, which in nature imply structure (genotype) and behavior (phenotype) differences. In EAs, identical genotypes produce the same fitness. Thus, a decrease in genotype diversity will necessarily cause a decrease in phenotype diversity. Hence, to measure entropy/diversity, one needs to define some structural measures.

Such measures, however, might be computational intensive in some instances of EAs (e.g., genetic programming) [2]. Fortunately, based on the described entropy/diversity relationship between genotype and phenotype, such measures at the phenotype level are sufficient. Figure 1 shows how the numbers and sizes of classes of a population affect entropy. High entropy in EAs reveals the presence of many unique fitness values, where the population is evenly distributed over those values, as shown in Figure 1(a). Figure 1(c) represents low entropy computed from a population which contains fewer unique fitness values as many individuals have the same fitness.

Rosca [10] calculates entropy for a population by first placing fitness values into fitness classes p_i and counting the size of each fitness class. The p_i is the proportion of the population occupied by the population partition i. Entropy is then defined as:

$$-\sum_i p_i \log_2 p_i. \tag{3}$$

This paper extends [10] to experiment with entropy, using different flexible cases of fitness classes, to facilitate explicit balance between exploration and exploitation.

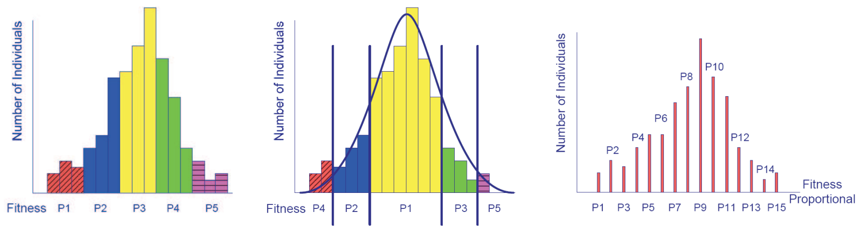


Figure 2. Linear (left), Gaussian (middle), and Fitness proportional (right).

Figure 2 shows three new cases for defining fitness classes:

- Linear: Assign a predefined yet changeable value to the number of fitness classes, n. For each generation, the interval between the best and worst fitness values is evenly partitioned into n sub-intervals as fitness classes (the left chart of Figure 2). An individual whose fitness value is occupied in a specific sub-interval is classified into the corresponding fitness class.

- Gaussian: The partition of fitness classes in this case is derived from Gaussian distribution. For each generation, fitness classes are 'spread out' from the average fitness value ($average$) with the standard deviation (σ). For example, the upper/lower bound of the first fitness class (P1 in the middle chart) is computed as $average \pm \sigma$. The boundaries of the successive classes (P2 – P5) can be generalized as $average \pm i\sigma$, where

$i \in \mathbb{Z}^+$ and $i \leq \frac{n}{2}$. For each generation, the lower bound of the leftmost fitness class is less than or equal to the smallest fitness value, and the upper bound of the rightmost fitness class is larger than or equal to the biggest fitness value.

- Fitness proportional: The fitness proportional approach is a variation of Rosca' approach [10]. Rosca's fitness classes are partitioned by individuals having the same phenotypes. p_i is the proportion of a population occupied in the i^{th} partition. In our approach, p_i is formalized as $\frac{f_i}{\Sigma_i^{Popsize} f_i}$, where f_i is the fitness value of an individual. p_i is the criterion for categorizing fitness classes. As all individuals of a population have different p_i (namely, different fitness values), the number of fitness classes n equals the population size ($Popsize$). If more than one individual contains the same fitness value (i.e., $p_i = p_j$, where $i \neq j$), $p_j \log_2 p_j$ is eliminated in the Eqn. (1) and $n < Popsize$. It is because two identical fitness classes are not necessary, and the elimination complies with the definition of diversity. Figure 2(c) shows 15 fitness classes sorted by p_i, and each of which has one or more individuals occupied.

4. Examples

Entropy driven exploration and exploitation have been experimented with on the suite of test functions presented in [15]. Due to lack of space only examples using the *Sphere Model* are presented in this section:

$$f(x) = \sum_i^d x_i^2, \tag{4}$$

where $x_i \in [-100, 100]$, d (dimension) $= 30$, and $\min(f) = f(0, \ldots, 0) = 0$. Best fitness value (B), Average fitness value (A), Worst fitness value (W), Population Diversity (D), Standard Deviation (S), Linear Entropy (E), Gaussian Entropy (G), Fitness Proportional Entropy (P), and Rosca Entropy (R) with respect to a population from generations 0 to 1500 (X-axis) are included in the following figures. Curves B, A, and W use the same definitions as all other EAs; curves E, G, P are defined in Section 3; curve S is the standard deviation of the fitness values of all individuals; curve D is the Euclidean distance between all individuals; and curve R is the entropy defined in [10]. All but entropy curves (E, G, P, and R) use the left Y-axis as the coordinate. The experimental results in the figures are the average values out of fifty rounds. Table 1 shows the initial values set up for the examples.

Figure 3 shows the results of a deterministic approach, which initializes $p_m = 0.11375$ and adjusts the value using the Fogarty formula [6]. In Figure 3, curves E, P, and R dramatically descend before generation 550. Curves B, A, W,

Table 1. Initial values of parameters in the following examples.

Parameter	Value	Parameter	Value
Maxgen	1500	Popsize	100
p_m	0.005	p_c	0.75
Epoch	50	Round	50

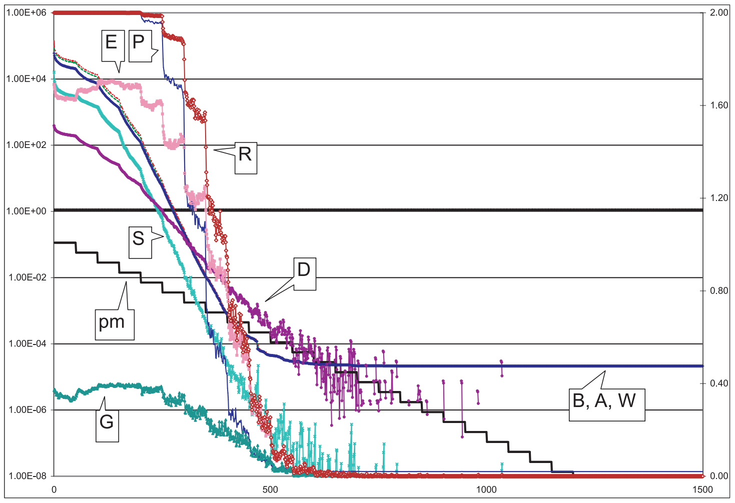


Figure 3. The Fogarty deterministic approach.

D, and S also precipitately decrease from generations 0 to 550. Such information indicates that the evolutionary process is inclined from more on exploration toward more on exploitation during this early phase. From generations 550 to 1035, more exploitation is applied than exploration such that curves B, A, W, S, and D become more and more flat. After generation 1035, the evolutionary process reaches the stable state which implies that no further process is necessary. The best value found using Fogarty deterministic approach is $2.13\,\mathrm{e}{-5}$.

Figure 4 presents the results using the $\frac{1}{5}$ success rule [9]. Such a rule determines p_m to be increased when the successful permutation rate (φ) is greater than $\frac{1}{5}$, and to be decreased when φ is less than $\frac{1}{5}$. In Figure 4, a good balance between exploration and exploitation (yet still more on exploration) can be found before generation 900: curves E and R are stable in the ranges between 1.4 and 1.65 and between 1.55 to 2.00, respectively; curves B, A, W, S, and D are smoothly decreased; and p_m is changed every 50 generations to adjust the mutation step. From generations 900 to 1220, curves E and R steeply decline, and curve G has downhill move. Such curves show that the evolutionary process is inclined from exploring to exploiting the current regions with relatively

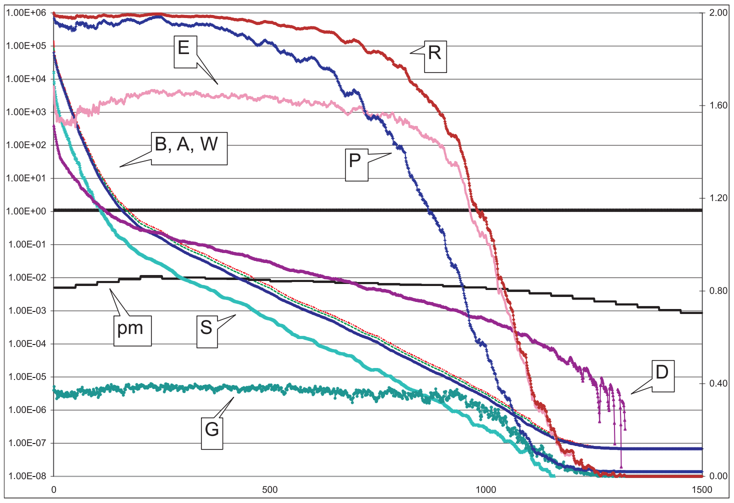


Figure 4. $\frac{1}{5}$ success rule approach.

small mutation steps. From generations 1220 to 1320, all entropy curves are getting flat and curve D has a ‘saw-toothed’ shape. Such curves imply that the searching process in the exploitation phase and is not stuck in local optima. The best value found using the $\frac{1}{5}$ success rule approach is $6.82\,\mathrm{e}{-8}$. Before exam-

```
init;
while ( t < Maxgen ) do
      callGA;
      if ( Entropy > 0.5 ) then
            pm := pm * 0.82
      fi;
      if ( Entropy < 0.5 ) then
            pm := pm * 1.22
      fi;
      t := t + Epoch
end;
```

Figure 5. PPCEA source code for an entropy-driven approach.

ining the last chart, an entropy-driven approach written in PPCEA is introduced in Figure 5. When entropy is greater than 0.5, p_m is decreased to facilitate the exploitation phase. Smaller mutation steps avoid the increase of population diversity. As entropy is smaller than 0.5, more exploration is required to avoid local optima. Therefore, p_m is increased to diversify the search regions. Such an example perfectly shows that balance between exploration and exploitation can be adjusted in synergy of entropy and p_m. Figure 6 shows the result using this source code.

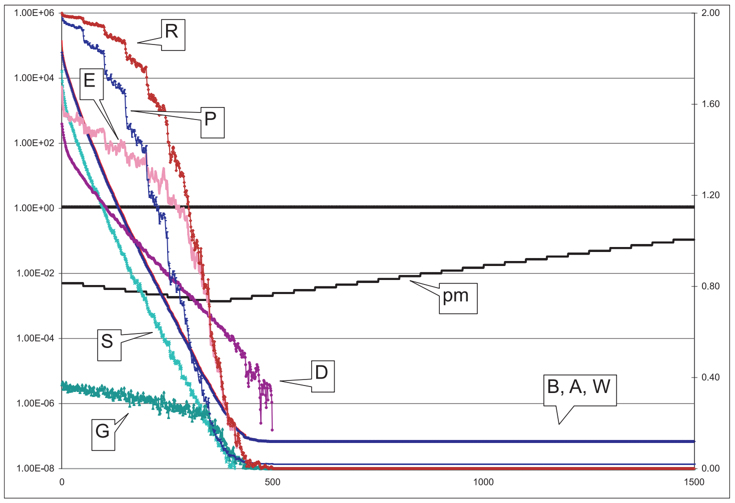


Figure 6. Entropy-driven approach.

In Figure 6, curves E, P, and R steeply decline between generations 0 and 450. Curves B, A, W, S, and D also diagonally traverse the plane. When curve E is between its midpoint (at generation 350) and upper bound (0.74 to 1.68), p_m is decreased (line 5 of Figure 5) to balance exploitation against exploration. As curve E is between its lower bound and midpoint (0 to 0.74), exploration outperforms exploitation by raising p_m. This phenomenon can be observed from curve D that declines more steeply and has a drastic “saw-toothed” shape from generations 400 to 500. Curve R is similar to curve E in terms of the shapes and the balance between exploration and exploitation. The best value found is the same as in the $\frac{1}{5}$ success rule. However, please note that the convergence is much better in the entropy-driven approach. Hence, many fitness evaluations over 500 generations can be skipped.

Figures 3, 4, and 6 also conclude that the linear and Rosca approaches for defining fitness classes are superior to Gaussian and fitness proportional ones in terms of providing the information for balancing exploration and exploitation.

5. Conclusion

The opinions on the research on exploration and exploitation are still widely divided [4]. In this paper, we introduce a novel entropy-driven exploration and exploitation approach. The balance between exploration and exploitation is fulfilled by the synergy of p_m, p_c and entropy on-line. The on-line adaptation mechanism involves PPCEA as to whether more exploitation or exploration is needed depending on the current progress of the algorithm and on the current estimated potential of discovering better solutions. The experimental results in Figures 3, 4, and 6 show that our approach can easily interpret the influence of exploration and exploitation using curve E and auxiliary curves.

References

[1] T. Bäck, D.B. Fogel, and Z. Michalewicz (eds.). *Handbook of Evolutionary Computation*. Oxford University Press, New York and Institute of Physics Publishing, Bristol, 1997.

[2] E. Burke, S. Gustafson, G. Kendall, and N. Krasnogor. Advanced Population Diversity Measures in Genetic Programming. *Lect. Notes Comput. Sc*, 2439:341–350, 2002.

[3] K. De Jong. The Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph.D. thesis, Department of Computer Science, University of Michigan, Ann Arbor, Michigan, 1975.

[4] A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter Control in Evolutionary Algorithms. *IEEE Trans. Evol. Comput.*, 3(2):124–141, 1999.

[5] A.E. Eiben and C.A. Schippers. On Evolutionary Exploration and Exploitation. *Fundamenta Informaticae*, 35(1-4):35–50, 1998.

[6] T.C. Fogarty. Varying the Probability of Mutation in the Genetic Algorithm. In *Proc. 3rd International Conference on Genetic Algorithms*, pages 104–109, 1989.

[7] J.J. Grefenstette. Optimization of Control Parameters for Genetic Algorithms. *IEEE Trans. Syst. Man Cyb.*, 16(1):122–128, 1986.

[8] S.-H. Liu, M. Mernik, and B.R. Bryant. Parameter Control in Evolutionary Algorithms by Domain-Specific Scripting Language PPCEA. In *Proc. International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004)*, pages 41–50, Ljubljana, Slovenia, 2004.

[9] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd edition. Springer-Verlag, 1996.

[10] J. Rosca. Entropy-Driven Adaptive Representation. In *Proc. Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 23–32, 1995.

[11] J.D. Schaffer et al. A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. In *Proc. 3rd International Conference on Genetic Algorithms*, pages 51–60, 1989.

[12] C. Shannon. A Mathematical Theory of Communication. *Bell Syst. Tech. J.*, 27:379–423, 623–656, 1948.

[13] R. Ursem. Diversity-Guided Evolutionary Algorithms. *Lect. Notes Comput. Sc.*, 2439:462–471, 2002.

[14] S. Whitehead. Learning from Delayed Rewards. Ph.D. thesis, King's College, Cambridge University, England, 1992.

[15] X. Yao, Y. Liu, and G. Lin. Evolutionary Programming Made Faster. *IEEE Trans. Evol. Comput.*, 3(2):82–102, 1999.

NICHING PROSPECTS

Mike Preuss
Department of Computer Science
University of Dortmund, Germany
mike.preuss@uni-dortmund.de

Abstract Although a large number of evolutionary algorithms have been proposed to efficiently treat multimodal problems, it is currently unclear under what conditions they can be faster than iterated local search algorithms. We tackle this question, assuming we had means to efficiently and errorlessly determine the corresponding basin of attraction for each individual (basin identification) by employing a simplified niching model EA that avoids superfluous local searches. Monte Carlo simulations show that outperforming the iterated local search is possible but difficult; the expected speedup is rather low if basins are approximately equally sized.

Keywords: Niching evolutionary algorithms, Basin model, Monte Carlo simulations

1. Introduction

Niching in *evolutionary algorithms* (EA) appears to be a heterogenous collection of techniques applied to enhance their ability to cope with multimodal objective functions by implementing some form of parallelization, either in terms of search space or time, or both. Does it contain all EA variants suggested for multimodal optimization? Surely not. But to state what exactly distinguishes niching approaches from other ones seems non-trivial, as—despite existing, partly contradictory definitions—the *evolutionary computation* (EC) community apparently does not yet possess a unified taxonomic view on multimodal evolutionary optimization. It is our aim to contribute to a movement into this direction by investigating what niching actually is and what it can do to improve evolutionary algorithms.

1.1 Niching Definitions

Out of the large set of publications dealing with niching or similar techniques in EC (e.g., De Jong [4] and Goldberg [7] as some of the earliest) we select only two opinions to show where to locate possible disagreements in defining niching. Mahfoud [12] gives the following functional specification of niching methods in an optimization context (p. 61):

The litmus test for a niching method, therefore, will be whether it possesses the capability to find multiple, final solutions within a reasonable amount of time, and to maintain them for an extended period of time.

He additionally states that the multiple solutions correspond to multiple local optimizers. Beyer et al. [6] include the process of separation, too. However, they also add diversity maintenance in their definition:

Niching—process of separation of individuals according to their states in the search space or maintenance of diversity by appropriate techniques, e.g., local population models, fitness sharing, or distributed EA

Whenever speaking of niches in EAs for static black box optimization, authors usually identify them with basins of attraction, at least for real-valued optimization. As Mahfoud points out, diversity maintenance is related to niching but must not be pursued too rigorously because it enables, but does not guarantee finding many basins, depending on the basin distribution within search space. In this sense, combining parts of both specifications, referring to basins of attraction, and leaving out diversity maintenance leads us to the following new definition:

Niching in EAs is a two-step procedure that a) concurrently or subsequently distributes individuals onto distinct basins of attraction and b) facilitates approximation of the corresponding (local) optimizers.

Undoubtedly, all EAs have local search capabilities. Therefore, it must be the process of locating basins that induces difficulties and requires experimentation with many EA variants to establish niching. In accordance to the explicit/implicit diversity maintenance scheme suggested by Eiben and Smith [5], we further partition niching EAs into two groups, performing explicit or implicit *basin identification*. Explicit basin identification methods—detecting the basin of each individual—divide the individuals into subpopulations, according to their basins.

1.2 Existing Approaches

Over the last 30 years, a large variety of niching techniques has been suggested. Comprehensive comparative studies are rare, but the existing (e.g., Mahfoud [11], Watson [23]) give hints on the relation between fitness landscape properties and performance of different niching methods. However, despite several recent approaches (Beasley et al. [2], Pétrowski [16, 17], Jelasity [8], Ursem [21], Wineberg [24], Li et al. [9], Streichert et al. [20], Shir [19], Ando et al. [1]), in the face of a multitude of possibilities one is tempted to resort to the ‘traditional’ methods *crowding* [4] and *sharing* [7], or variations thereof. What is the reason for this dissatisfactory tentativeness? There may be several, e.g.:

- The diverse character of the proposed methods, algorithmically as well as in descent from different origins, complicates gathering a viable overview. Available results are not directly comparable.
- Only few taxonomic attempts exists for multimodal EAs, and the existing ones by Ursem [22], and Eiben and Smith [5] utilize different, mutually incomparable criteria, as avoid/repair strategy in the former and diversity maintenance in the latter case.
- The aspired task is not concrete enough or unreachable.

As the stream of new methods does not cease, one may ask what the motivation behind designing new niching EAs is. The seemingly underlying, yet unreached aim is to convincingly beat one of the simplest algorithms for multimodal objective functions, the iterated/parallelized hillclimber/local search. According to the NFL, this task is venturous when optimizing general multimodal problems, but it may be possible for problem classes exhibiting certain exploitable properties.

1.3 Biological Background

Importing concepts from biology (ecology), which undoubtedly is the origin of the general idea of niching for EAs, appears problematic. Biologists now tend to view separation into niches as a process the affected living beings actively take part in, also treated as *niche construction*, Odling-Smee et al. [15]. Whereas individuals in canonical EAs are merely collections of values without a ‘life of their own’, living beings act on highly dynamic fitness landscapes and must pursuit several objectives (e.g., food and reproduction).

The related problem of speciation—the term species often denotes separate subpopulations in niching EAs—currently is one of the most progressive research topics in evolutionary biology, with Mayr’s reproductively isolated populations [13, 14] and the allopatric (geographic) speciation mechanism as predominant concepts. Although these two can be (and are) adapted for use in EAs, biologists are still far from having reached consensus concerning all issues raised with the problem of speciation, and thus not able to provide a proper foundation to argue on in the EC field. The current state of the speciation debate is summarized in Coyne and Orr [3]. As an example for a controversially discussed yet unsolved problem, we name the formation and maintenance of sexual reproduction. This issue is dismissed in EA research, in favor of asexual populations, for which in turn no widely accepted speciation concept exists in biology. In consequence, biological terms shall be used with extreme care when applied to niching EAs to prevent conceiving meanings where there are only metaphors.

2. Aims and Methods

In the following, our main task is to gather evidence in favor of or against the (in EC) prevalent belief that niching EAs can outperform iterated local search (ILS, see Lourenco et al. [10]) algorithms. Note that this is an existential pre-condition for designing further niching EAs as these are usually algorithmically much more complex. We thus do the second step prior to the first and simply assume the existence of an efficient basin identification method for population based EAs. This would enable deciding if any two individuals are located in the same basin or not. The first question to investigate thus is:

- Given that basin identification works, how much faster can a niching EA be in terms of a *redundancy factor* (measuring superfluous local searches, see Beasley et al. [2]), compared to ILS algorithms?

We employ a very simple niching model EA and estimate the amount of local searches needed for reasonable basin numbers and population sizes by means of Monte Carlo simulations.

3. Simplistic Niching Model EA

Modeling the behavior of a real niching EA on an idealized multimodal objective function still bears enormous complexity. The whole local search process in the detected basins must be considered, and is likely to heavily depend on algorithm and problem parameters.

Hence, for our niching model EA, we choose the single local search as unit of measurement. We further assume that for any (start) population of search points, a basin identification method exists that returns an errorless search point to basin mapping in negligible time. This condition describes an optimal situation—for any real niching EA, basin identification will require computational effort. Additionally, it may not be possible to detect the basin of an individual as soon as it enters it. Thus, the implied advantage of our ideal niching EA which consists of breaking unnecessary local searches at the start may not be realizable in full. But, unless other techniques are applied to reduce the optimization effort (e.g., utilization of attained information to speed up subsequent local searches), *any* niching EA can not be faster in terms of local searches than the niching model EA—we obtain an estimation for a lower bound.

In a real niching EA, the number of covered c of a total of b basins for a randomly initialized start population would fluctuate according to population size and basin distribution. However, we set it constant to simplify studying

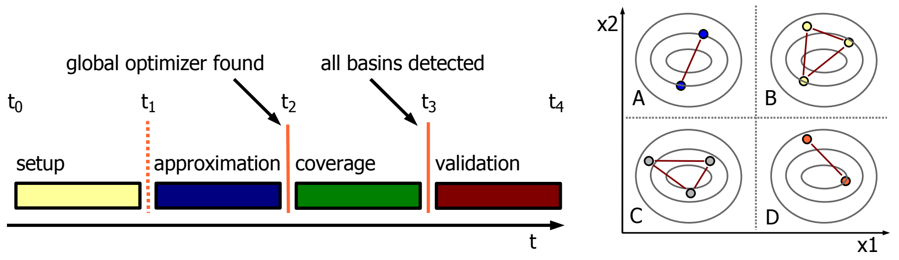


Figure 1. Left: Four phases of a heuristic optimization process. We are interested in detecting t_2 and t_3. Right: Niching model EA population after initialization and basin identification. Individuals residing in one basin are connected by lines.

the effect of saving local searches. Summarizing, the model is based on the following assumptions:

- Basin identification is perfect and has zero cost.
- Local searches always succeed and have equal cost of 1.
- Any start population covers exactly c of b existing basins.

Without basin identification, one is thrown back to iterated/parallelized local searches for which the required effort is known [2]. Covering the whole basin set with randomly initialized local searches results in a relative local search overhead, measured by the redundancy factor R:

$$R = \sum_{i=1}^{b} \frac{1}{i} \overset{b>3}{\approx} \gamma + \ln b. \tag{1}$$

For entering each of the b basins at least once, $R \times b$ local searches are necessary on average. Here, $\gamma \approx 0.577$ is the Euler-Mascheroni constant.

Instead of conducting single local searches, the niching model EA repeatedly starts with a randomly initialized set of individuals and performs only necessary local searches until all basins have been visited (Figure 2). We do not specify how the local searches are implemented; they may be realized e.g. by mating restrictions, or separate populations, or embedded helper methods. Note that basin identification only needs to detect if individuals are located in the same basin; it is not required to properly recognize each basin as such (Figure 1, right).

What kind of performance data do niching model EA simulation runs deliver? Figure 1 (left) displays the phases of any heuristic optimization algorithm in terms of basin detection. During setup, the algorithm is prepared and started and yields the first result in t_1. This approximation phase lasts until the global optimizer is hit the first time at t_2. It shall be noted that especially in real-world applications, this point is often never reached because evaluations may be too

```
niching model EA {
    repeat {
        randomly initialize individuals on $c$ of $b$ basins;
        basin identification: match individuals to basins;
        select one individual per basin $=$ $c$ individuals;
        perform $c$ local searches on selected individuals;
    } until stopped externally (all basins visited);
}
```

Figure 2. Niching model EA in pseudo-code.

costly. The coverage phase is needed to visit each basin at least once and ends with t_3. Unless the number of basins is known in advance, it seems impossible to determine t_3 from inside an optimization algorithm. It is up to the user to stop it when no new information can be obtained from running further (t_4). In case of the niching model EA, t_2 and t_3 are measurable because the basin set is known. Note that the redundancy factor stated in Eqn. (1) is equivalent to t_3 which thus not refers to the expected first hitting time but to the end of the coverage phase.

In the following, we present two experiments in order to investigate the influence of basin number b and coverage/parallelized searches c on t_2 and t_3 for the niching model EA. Firstly, equally sized basins are studied. Secondly, we review occuring changes for unequally sized basins.

Experiment 1: Global optimizer/coverage detection times, equal basins.

Pre-experimental planning: The appropriate number of repeats is determined to 10,000 during first tests; relative standard deviations are thus decreased well below 1 %.

Task: Measure t_2 and t_3 and detect how they relate to the number of basins b and parallel searches (covered basins) c.

Setup: We simulate all $b, c \in \{1, 2, \dots, 50\} : b \geq c$ with 10,000 repeats each. Probabilities for encountering any basin during random initialization are equal and set to $\frac{1}{b}$.

Experimentation/Visualization: Figure 3 depicts averaged measures for t_3 (left) and t_2 (right).

Observations: Whereas the number of parallel searches c clearly affects t_3, it seems to lack any influence on t_2 which only depends on the number of basins b ($\widehat{\mathbb{E}}(t_2) = b$). To clarify the influence of c on t_3, we picture measured t_3, divided by the approximation given by Eqn. (1) (Figure 4).

Discussion: Different values for c do not change t_2 at all, meaning that parallel searches do not increase or decrease the expected time needed to arrive at the

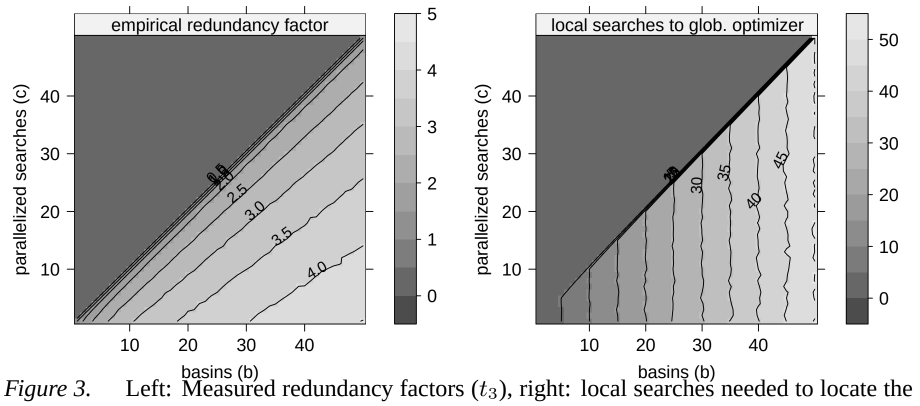


Figure 3. Left: Measured redundancy factors (t_3), right: local searches needed to locate the global optimum (t_2). Both are averaged from 10,000 simulations per point.

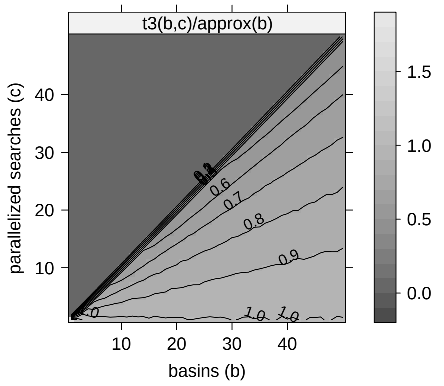


Figure 4. Measured redundancy factor (t_3) as fraction of the approximation for repeated single local searches (see Eqn. (1)) for the same number of basins b. If $\frac{c}{b} \leq 0.9$, the coefficient of the observed linear relation is similar to $\sqrt[3]{1-\frac{c}{b}}$ (by visual comparison), resulting in the approximation $t_3(b,c) \approx \sqrt[3]{1-\frac{c}{b}}(\gamma + \ln b)$.

global optimum. Approaches targetting at this effect for approximately equally sized basins are thus doomed to fail. Nevertheless, the amount of local searches needed for complete coverage (t_3) is reduced for $c > 1$. However, the save is small and the utilized basin identification technique must be very efficient not to loose it again.

Experiment 2: Detect t_2 and t_3 for unequally sized basins.

Pre-experimental planning: The maximum size difference was fixed to 10 as first experiments indicate a sufficient change in obtained results.

Task: Similar to Exp. 1.

Setup: Similar to Exp. 1, but with uniform randomly distributed relative basin sizes between 1.0 and 10.0.

Experimentation/Visualization: Averaged t_2 and t_3 measures are depicted in Figure 5.

Observations: Firstly, measured values for t_3 arrive at much higher values than for the case of equal basins. Secondly, the growth rate on the basin axis

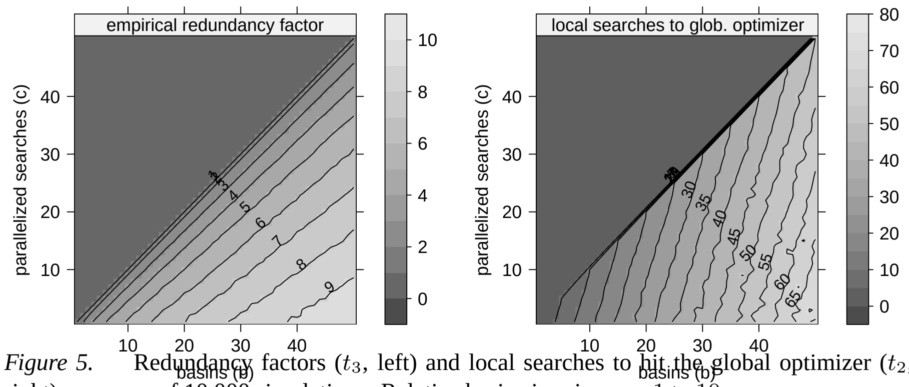


Figure 5. Redundancy factors (t_3, left) and local searches to hit the global optimizer (t_2, right), averages of 10,000 simulations. Relative basin size sizes are 1 to 10.

($c = 1$) appears to be between logarithmic and linear—compared to logarithmic in Exp. 1. In contrast to the findings of Exp. 1, t_2 now is affected by changing values of c. For $b = c$, that is, all basins are covered by the parallel search, $t_2 = b$ still holds. But the lower $\frac{c}{b}$ is, the larger t_2 gets.

Discussion: Obviously, optimization gets harder if basins are unequally sized. This is well in accordance with expectation. Now, t_2 and t_3 both depend on $\frac{c}{b}$. We may conclude that larger relative basin size differences lead to larger potential performance advantages of niching EAs. On the other hand, basin identification probably gets harder, too.

4. Conclusions

Previous studies (e.g., Preuss et al. [18]) have shown that canonical EAs are not well suited for multimodal optimization. Are niching EAs? According to our simulations, the is some exploitable potential, but it is small for equally sized basins. It appears that chances are getting better the larger basin size differences are. However, we assumed existence of an efficient basin identification method, which utilizes population topologies in search space and thus depends on the number of dimensions of a problem. Whether and for what problems such technique can be fast enough to enable outperforming an ILS still remains to be seen.

References

[1] S. Ando, E. Suzuki, and S. Kobayashi. Sample-based Crowding Method for Multimodal Optimization in Continuous Domain. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2005)*, Edinburgh, UK, 2005.

[2] D. Beasley, D.R. Bull, and R.R. Martin. A sequential niche technique for multimodal function optimization. *Evol. Comput.*, 1(2):101–125, 1993.

[3] J.A. Coyne and H.A. Orr. *Speciation*. Sinauer Associates, Inc., Sunderland, MA, 2004.

[4] K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

[5] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin, Heidelberg, New York, 2003.

[6] H.-G. Beyer, E. Brucherseifer, W. Jakob, H. Pohlheim, B. Sendhoff, and T.B. To. Evolutionary algorithms – terms and definitions. VDI/VDE guideline 3550, leaf 3, 2003. `ls11-www.cs.uni-dortmund.de/people/beyer/EA-glossary/def-engl-html.html`.

[7] D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proc. 2nd International Conference on Genetic Algorithms*, pages 41–49, Mahwah, NJ, USA, 1987.

[8] M. Jelasity. Uego, an abstract niching technique for global optimization. *Lect. Notes Comput. Sc.*, 1498:378–387, 1998.

[9] J.-P. Li, M.E. Balazs, G.T. Parks, and P.J. Clarkson. A species conserving genetic algorithm for multimodal function optimization. *Evol. Comput.*, 10(3):207–234, 2002.

[10] H.R. Lourenco, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer, 2002.

[11] S.W. Mahfoud. A comparison of parallel and sequential niching methods. In *Proc. 6th International Conference on Genetic Algorithms*, pages 136–143, San Francisco, CA, USA, 1995.

[12] S.W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, IL, 1995.

[13] E. Mayr. *Systematics and the Origin of Species*. Columbia University Press, New York, 1942.

[14] E. Mayr. Species, classification, and evolution. In R. Arai, M. Kato, and Y. Doi, editors, *Biodiversity and Evolution*. National Science Museum Foundation, Tokyo, 1995.

[15] F.J. Odling-Smee, K.N. Laland, and M.W. Feldman. *Niche Construction—The neglected process in evolution*. Princeton University Press, Princeton and Oxford, 2003.

[16] A. Pétrowski. A clearing procedure as a niching method for genetic algorithms. In *Proc. IEEE International Conference on Evolutionary Computation (ICEC 1996)*, pages 798–803, Nagoya, Japan, 1996.

[17] A. Pétrowski and M.G. Genet. A Classification Tree for Speciation. In *Proc. IEEE Congress on Evolutionary Computation (CEC 1999)*, pages 204–211, Washington, DC, USA, 1999.

[18] M. Preuss, L. Schönemann, and M. Emmerich. Counteracting genetic drift and disruptive recombination in $(\mu \overset{+}{,} \lambda)$-ea on multimodal fitness landscapes. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2005)*, pages 865–872, New York, NY, USA, 2005.

[19] O. M. Shir and T. Bäck. Niching in evolution strategies. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2005)*, pages 915–916, New York, NY, USA, 2005.

[20] F. Streichert, G. Stein, H. Ulmer, and A. Zell. A clustering based niching method for evolutionary algorithms. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 644–645, Chicago, IL, USA, 2003.

[21] R. K. Ursem. Multinational evolutionary algorithms. In *Proc. IEEE Congress on Evolutionary Computation (CEC 1999)*, volume 3, pages 1633–1640, Washington, DC, USA, 1999.

[22] R. K. Ursem. *Models for Evolutionary Algorithms and Their Applications in System Identification and Control Optimization*. PhD thesis, EVALife, Department of Computer Science, University of Aarhus, 2003.

[23] J.-P. Watson. A performance assessment of modern niching methods for parameter optimization problems. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 1999)*, volume 1, pages 702–709, Orlando, FL, USA, 1999.

[24] M. Wineberg. *Improving the Behavior of the Genetic Algorithm in a Dynamic Environment*. PhD thesis, Carleton University, Ottawa, Canada, 2000.

CONTROL PARAMETERS IN SELF-ADAPTIVE DIFFERENTIAL EVOLUTION

Janez Brest, Viljem Žumer, Mirjam Sepesy Maučec
Faculty of Electrical Engineering and Computer Science
University of Maribor, Slovenia
{janez.brest,zumer,mirjam.sepesy}@uni-mb.si

Abstract In this paper we present experimental results to show deep view on how self-adaptive mechanism works in differential evolution algorithm. The results of the self-adaptive differential evolution algorithm were evaluated on the set of 24 benchmark functions provided for the CEC2006 special session on constrained real parameter optimization. In this paper we especially focus on how the control parameters are being changed during the evolutionary process.

Keywords: Control parameters, Differential evolution, Self-adapting

1. Introduction

Differential Evolution (DE) [8, 9, 10, 13, 14, 15, 16] has been shown to be a powerful evolutionary algorithm for global optimization in many real problems [11, 12]. Although the DE algorithm has been shown to be a simple yet powerful evolutionary algorithm for optimizing continuous functions, users are still faced with the problem of preliminary testing and hand-tuning of the evolutionary parameters prior to commencing the actual optimization process [16].

Different problems usually require different setting for the control parameters. Self-adaptation allows an evolution strategy to adapt itself to any general class of problems by reconfiguring itself accordingly, and to do this without any user interaction [1, 2, 6]. Based on the experiment in [4], the necessity of changing control parameters during the optimization process is also confirmed. In literature, self-adaptation is usually applied to the F and CR control parameters [3, 4].

In our previous paper [5] the performance of the self-adaptive differential evolution algorithm was evaluated on the set of 24 benchmark functions provided for the CEC2006 special session on constrained real parameter optimization [7]. The method in [5] extended individuals that have not only decision variables but also control parameters F and CR, where F is a scaling factor and CR is a crossover rate. These parameters are changed/optimized by DE, too. The authors utilized the lexicographic ordering, in which the constraint violation precedes the objective function, to solve constrained problems.

In this paper we investigate how these parameters adapt during search for some of the test functions (i.e. some typical runs). Do they really change much and how?

The main focus in this paper is related with our previous paper [5] where the performance of the self-adaptive differential evolution algorithm was evaluated on the set of 24 benchmark functions [7]. In [5] results are presented, how efficient our self-adaptive DE algorithm is on constraint-base optimization. In this paper we focus only on a self-adapting control parameters. We want to answer, how the control parameter are being changed during the evolutionary process.

2. Background

In this section we give overview of previous works, which gives the basis of this paper. The original differential evolution (DE) algorithm is briefly presented. Then the self-adaptive mechanism used in our DE algorithm is outlined.

2.1 The Differential Evolution Algorithm

DE creates new candidate solutions by combining the parent individual and several other individuals of the same population. A candidate replaces the parent only if it has better fitness value. DE has three parameters: amplification factor of the difference vector, F, crossover control parameter, CR, and population size, NP.

The population of the original DE algorithm [13, 14, 15] contains NP D-dimensional vectors:

$$\vec{x}_{i,G} = \{x_{i,1,G}, x_{i,2,G}, \ldots, x_{i,D,G}\},\ i = 1, 2, \ldots, NP.$$

G denotes the generation. During one generation for each vector, DE employs the mutation and crossover operations to produce a trial vector:

$$\vec{u}_{i,G} = \{u_{i,1,G}, u_{i,2,G}, \ldots, u_{i,D,G}\},\ i = 1, 2, \ldots, NP.$$

Then a selection operation is used to choose vectors for the next generation $(G + 1)$.

The initial population is selected randomly in a uniform manner between the lower ($x_{j,low}$) and upper ($x_{j,upp}$) bounds defined for each variable x_j. These bounds are specified by the user according to the nature of the problem. After initialization, DE performs several vector transforms (operations) in a process called evolution.

2.2 Mutation Operation

Mutation for each population vector creates a mutant vector:

$$\vec{x}_{i,G} \Rightarrow \vec{v}_{i,G} = \{v_{i,1,G}, v_{i,2,G}, \ldots, v_{i,D,G}\},\ i = 1, 2, \ldots, NP.$$

Mutant vector can be created by using one of the mutation strategies. There are many original DE strategies. The strategies used in this paper are:

- 'rand/1': $\vec{v}_{i,G} = \vec{x}_{r_1,G} + F \cdot (\vec{x}_{r_2,G} - \vec{x}_{r_3,G})$,
- 'current to best/1':
  $\vec{v}_{i,G} = \vec{x}_{i,G} + F \cdot (\vec{x}_{best,G} - \vec{x}_{i,G}) + F \cdot (\vec{x}_{r_1,G} - \vec{x}_{r_2,G})$,
- 'rand/2':
  $\vec{v}_{i,G} = \vec{x}_{r_1,G} + F \cdot (\vec{x}_{r_2,G} - \vec{x}_{r_3,G}) + F \cdot (\vec{x}_{r_4,G} - \vec{x}_{r_5,G})$,

where the indexes r_1, r_2, r_3, r_4, r_5 represent the random and mutually different integers generated within range $[1, NP]$ and also different from index i. F is a mutation scale factor within the range $[0, 2]$, usually less than 1. $\vec{x}_{best,G}$ is the best vector in generation G.

2.3 Crossover Operation

After mutation, a 'binary' crossover operation forms the final trial vector, according to the i-th population vector and its corresponding mutant vector.

$$u_{i,j,G} = \begin{cases} v_{i,j,G} & \text{if } rand(0,1) \le CR \text{ or } j = j_{rand}, \\ x_{i,j,G} & \text{otherwise} \end{cases}$$

$$i = 1, 2, \ldots, NP \text{ and } j = 1, 2, \ldots, D.$$

CR is a crossover parameter or factor within the range $[0, 1)$ and presents the probability of creating parameters for trial vector from a mutant vector. Index j_{rand} is a randomly chosen integer within the range $[1, NP]$. It is responsible for the trial vector containing at least one parameter from the mutant vector.

2.4 Selection Operation

The selection operation selects according to the fitness value of the population vector and its corresponding trial vector, which vector will survive to be a member of the next generation. For example, if we have a minimization problem, we will use the following selection rule:

$$\vec{x}_{i,G+1} = \begin{cases} \vec{u}_{i,G} & \text{if } f(\vec{u}_{i,G}) < f(\vec{x}_{i,G}), \\ \vec{x}_{i,G} & \text{otherwise.} \end{cases}$$

2.5 The Self-Adaptive Differential Evolution Algorithm

In [4] a self-adaptive control mechanism was used to change the control parameters F and CR during the run.

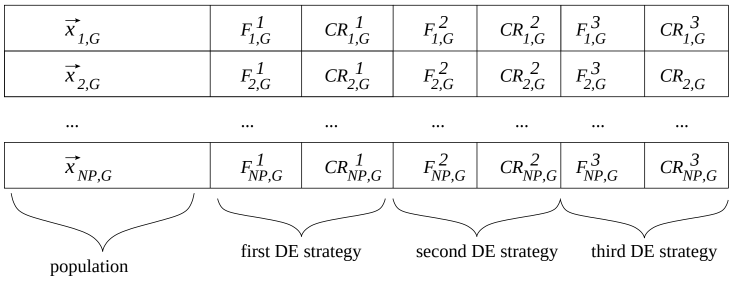

$\vec{x}_{1,G}$ $F^1_{1,G}$ $CR^1_{1,G}$ $F^2_{1,G}$ $CR^2_{1,G}$ $F^3_{1,G}$ $CR^3_{1,G}$
$\vec{x}_{2,G}$ $F^1_{2,G}$ $CR^1_{2,G}$ $F^2_{2,G}$ $CR^2_{2,G}$ $F^3_{2,G}$ $CR_{2,G}$
... ...
$\vec{x}_{NP,G}$ $F^1_{NP,G}$ $CR^1_{NP,G}$ $F^2_{NP,G}$ $CR^2_{NP,G}$ $F^3_{NP,G}$ $CR^3_{NP,G}$
population
first DE strategy
second DE strategy
third DE strategy

Figure 1. Self-adapting: encoding aspect of three DE strategies.

Figure 1 shows a solution how the control parameters of three original DE's strategies are encoded in each individual. Each strategy uses its own control parameters. The solution to apply even more strategies into our algorithm is straight-forward. New control parameters $F^k_{i,G+1}$ and $CR^k_{i,G+1}$, $k = 1, 2, 3$, were calculated as follows:

$$F^k_{i,G+1} = \begin{cases} F_l + rand_1 * F_u & \text{if } rand_2 < \tau_1, \\ F_{i,G} & \text{otherwise,} \end{cases}$$

$$CR^k_{i,G+1} = \begin{cases} rand_3 & \text{if } rand_4 < \tau_2, \\ CR_{i,G} & \text{otherwise.} \end{cases}$$

and they produce control parameters F and CR in a new parent vector. k represents selected DE strategy. When a new parent vector is calculated, only one strategy is active. In each iteration one strategy is chosen to be active. $rand_j, j \in \{1, 2, 3, 4\}$ are uniform random values within the range $[0, 1]$. τ_1 and τ_2 represent probabilities to adjust control parameters F and CR, respectively. τ_1, τ_2, F_l, F_u were taken fixed values $0.1, 0.1, 0.1, 0.9$, respectively. The new F takes a value from $[0.1, 1.0]$, and the new CR from $[0, 1]$ in a random manner. $F_{i,G+1}$ and $CR_{i,G+1}$ are obtained before the mutation is performed. So they influence the mutation, crossover and selection operations of the new vector $\vec{x}_{i,G+1}$.

In experiments in [5], the proposed jDE-2 algorithm uses the following three strategies 'rand/1/bin', 'current to best/1/bin', and 'rand2/bin'. The first pair of self-adaptive control parameters F and CR belongs to the 'rand/1/bin' strategy and the second pair belongs to 'current to best/1/bin' strategy, etc. The population size NP was set to 200. The maximal number of function evaluations (FES) is 500,000 for all benchmark functions.

The algorithm distinguishes between feasible and infeasible individuals: any feasible solution is better than any infeasible one.

The jDE-2 algorithm was tested on 24 CEC2006 special session benchmark functions. For 22 functions the jDE-2 algorithm has successfully found feasible solution. For $g20$ and $g22$ functions no feasible solution has been found.

3. Experimental Results

In this section we present results of experiments, which were made in order to find an answer, how the control parameters are adapted during evolutionary process.

In self-adaptive DE, F and CR values are being changed during evolutionary process. If we want to look into evolutionary process, we should look at fitness values.

In Figures 2–4 F and CR values of the active strategy are depicted for the selected set of functions $g01$, $g02$, $g05$, $g07$, $g10$, $g14$, $g15$, $g16$, $g17$, $g18$, $g19$, $g20$. A dot is plotted only when the best fitness value in generation is improved.

The values of control parameter F and CR for function $g01$ are quite equally distributed, F takes value from the $[0.1, 1]$ and CR from the $[0, 1]$.

For function $g02$ the values of control parameter F are in most cases less or equal 0.5 in the first 200,000 evaluations. After that F values are predominantly greater than 0.5. The values of control parameter CR are near 1 mostly.

Sometimes algorithm solves test problem before reaching the maximal number of FES, therefore some graphs (e.g., functions $g01$, $g10$, etc.) have not dots for all FES.

It can be seen that the graphs differ from each other to a great extend. It is difficult to obtain (general) one set of control parameter values, which will fit the best for all benchmark problems.

In the additional experiment, we run our algorithm without self-adaptation. The values of control parameters were $F = 0.5$ and $CR = 0.9$, and they were fixed during evolutionary process.

The algorithm with self-adaptation performs 11 % better than algorithm with fixed control parameters. The detailed performance results of our self-adaptive algorithm are in [5].

4. Conclusions

This paper shows that the DE control parameters F and CR changed (adapted) their values during evolutionary process. For selected CEC2006 benchmark functions the graphs of F and CR values during the evolution process are presented in the paper.

The experimental results confirm the hypothesis that the best setting for control parameters is problem dependent.

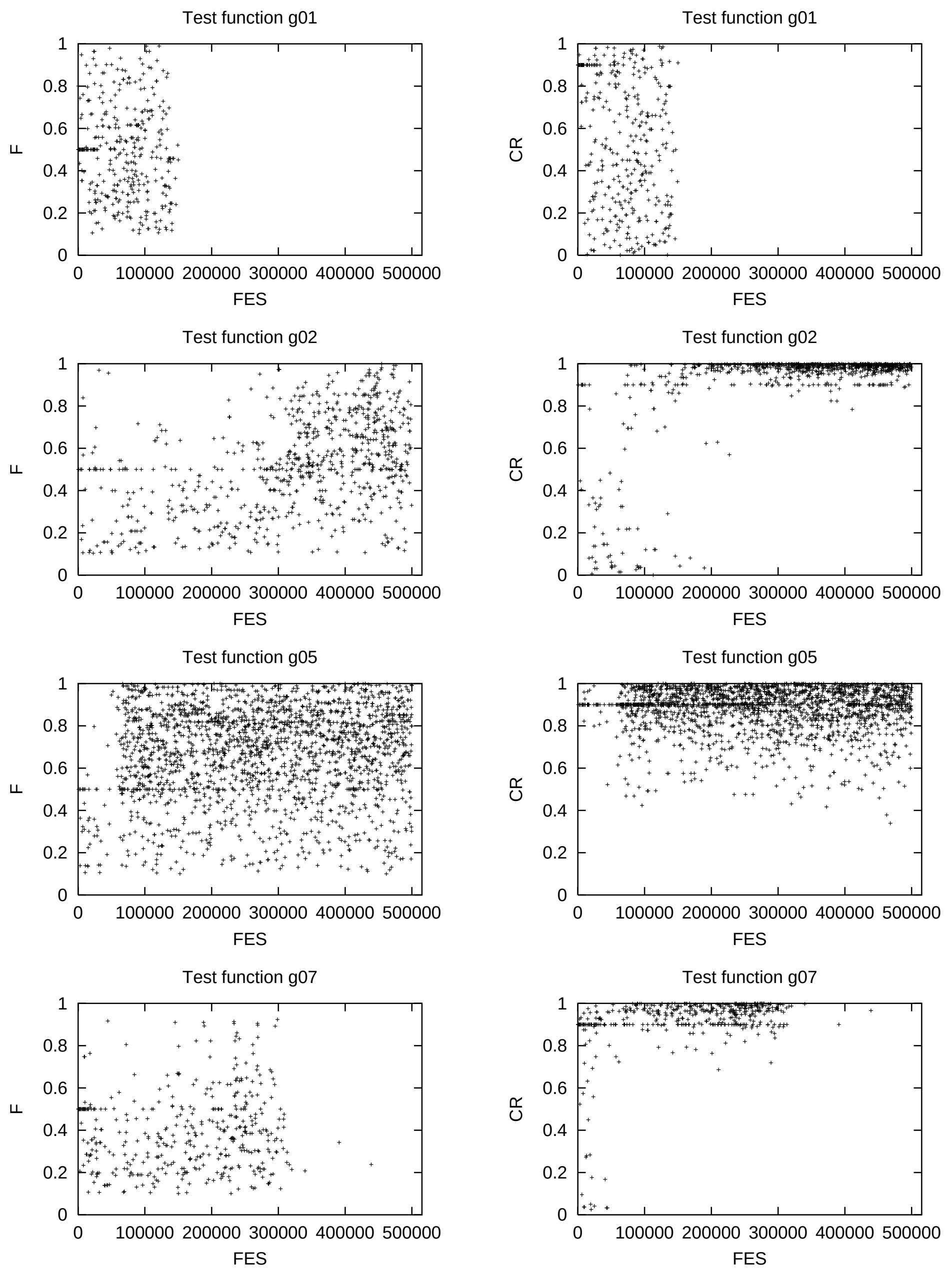


Figure 2. F and CR values for functions $g01$, $g02$, $g05$, and $g07$.

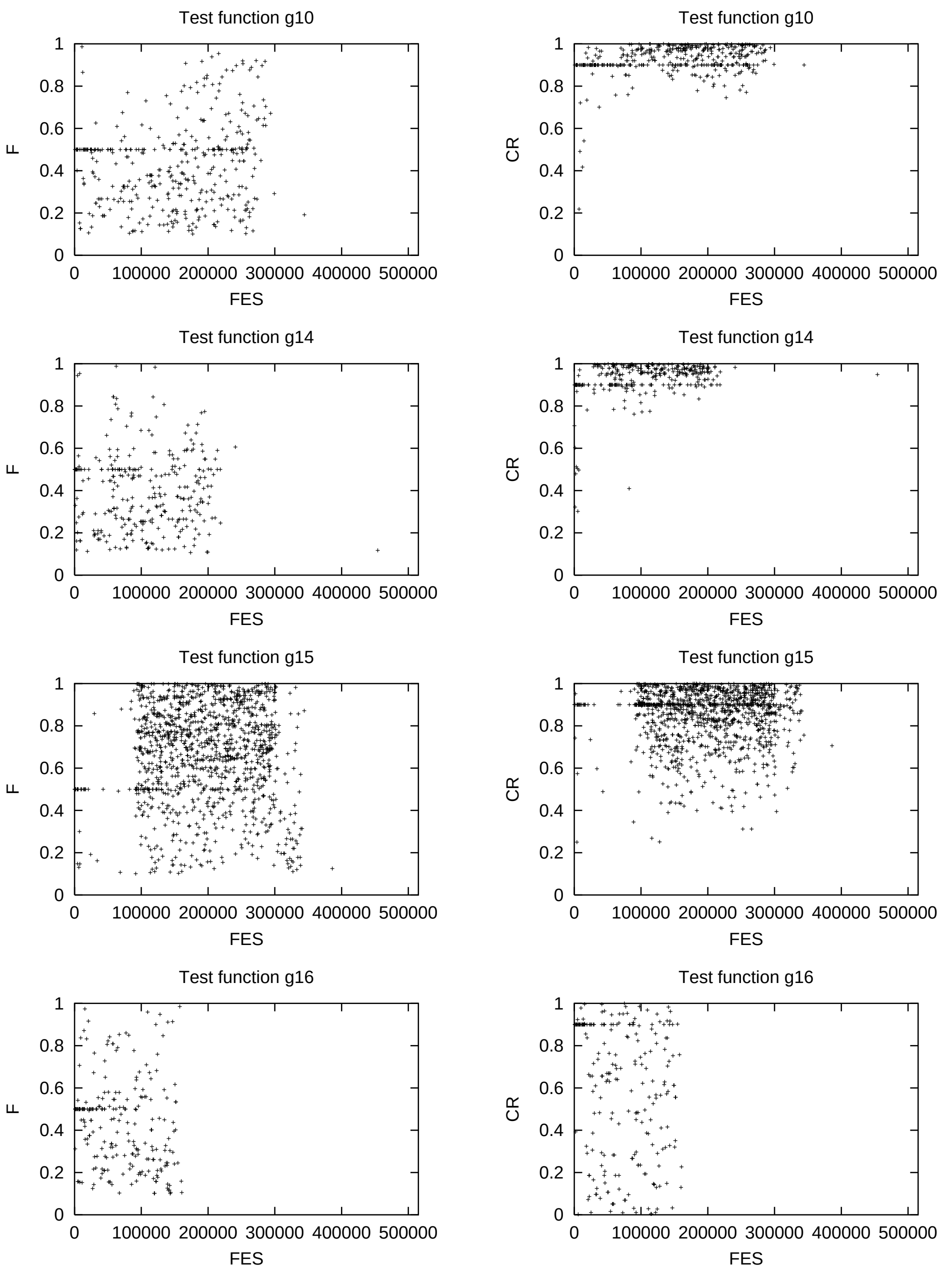


Figure 3. F and CR values for functions $g10$, $g14$, $g15$, and $g16$.

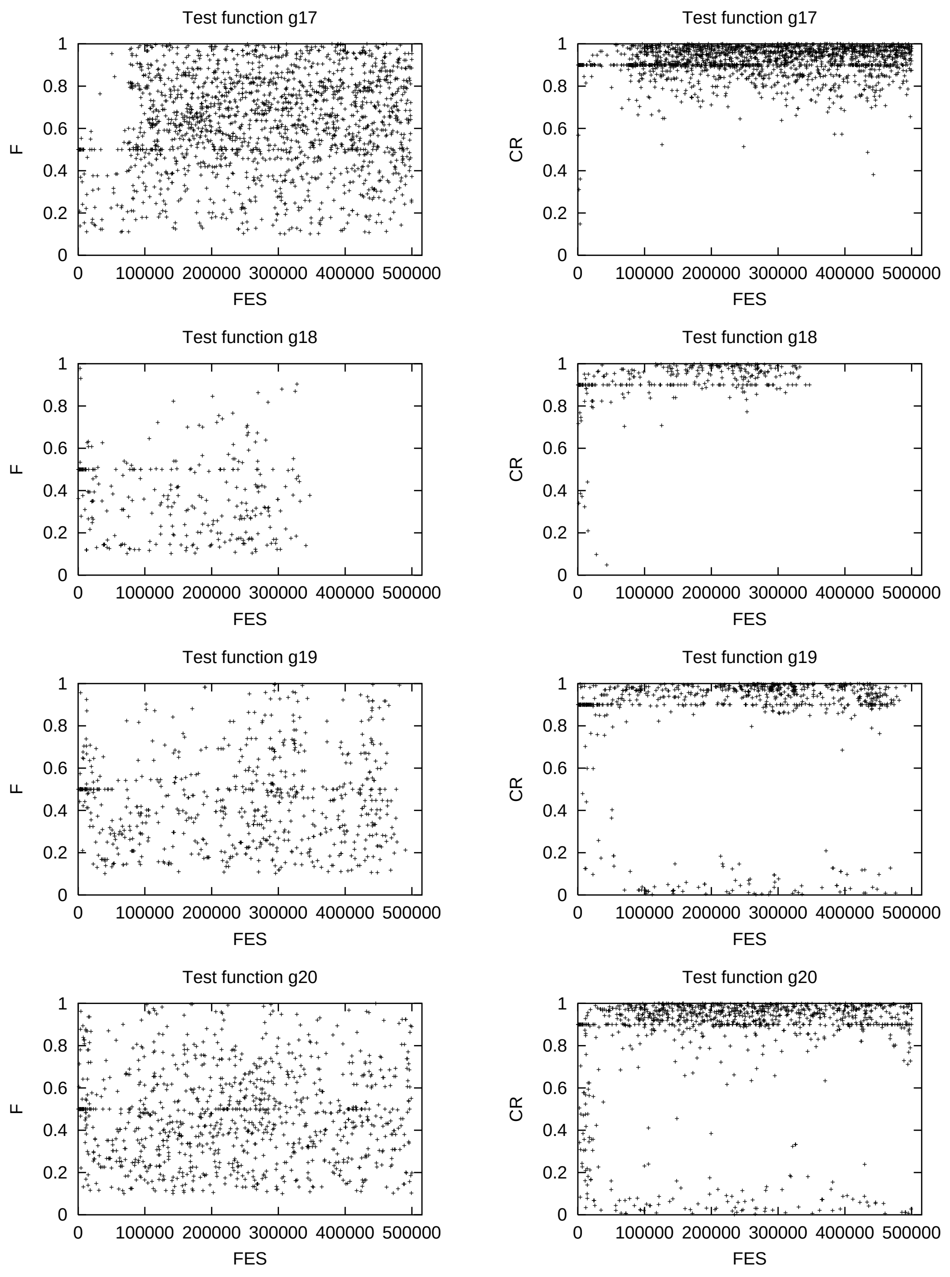


Figure 4. F and CR values for functions $g17$, $g18$, $g19$, and $g20$.

In this paper we used three DE strategies. The analysis how the control parameters are changed in particularly DE strategy is a challenge for the future work.

References

[1] T. Bäck. Adaptive Business Intelligence Based on Evolution Strategies: Some Application Examples of Self-Adaptive Software. *Infor. Sc.*, 148(1-4):113–121, 2002.

[2] T. Bäck, D. B. Fogel, and Z. Michalewicz (eds.). *Handbook of Evolutionary Computation*. Oxford University Press, New York and Institute of Physics Publishing, Bristol, 1997.

[3] J. Brest, B. Bošković, S. Greiner, V. Žumer, and M. Sepesy Maučec. Performance Comparison of Self-Adaptive and Adaptive Differential Evolution Algorithms. *Soft Comput.*, 2006. To appear.

[4] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. *IEEE Trans. Evol. Comput.*, 2006. To appear.

[5] J. Brest, V. Žumer, and M. Sepesy Maučec. Self-adaptive Differential Evolution Algorithm in Constrained Real-Parameter Optimization. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2006)*, pages 215–222, Vancouver, BC, Canada, 2006.

[6] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Natural Computing. Springer-Verlag, Berlin, 2003.

[7] J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, N. Suganthan, C. A. C. Coello, and K. Deb. Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization. Report #2006005, Nanyang Technological University, Singapore, 2005. `www.ntu.edu.sg/home/EPNSugan`.

[8] J. Liu and J. Lampinen. Adaptive Parameter Control of Differential Evolution. In *Proc. 8th International Conference on Soft Computing*, pages 19–26, Brno, Czech Republic, 2002.

[9] J. Liu and J. Lampinen. On Setting the Control Parameter of the Differential Evolution Method. In *Proc. 8th International Conference on Soft Computing*, pages 11–18, Brno, Czech Republic, 2002.

[10] J. Liu and J. Lampinen. A Fuzzy Adaptive Differential Evolution Algorithm. *Soft Comput.*, 9(6):448–462, 2005.

[11] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, Berlin, 2000.

[12] Z. Michalewicz and M. Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evol. Comput.*, 4(1):1–32, 1996.

[13] J. Rönkkönen, S. Kukkonen, and K. V. Price. Real-Parameter Optimization with Differential Evolution. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2005)*, vol. 1, pages 506–513, Edinburg, UK, 2005.

[14] R. Storn and K. Price. Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, Berkeley, CA, 1995.

[15] R. Storn and K. Price. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *J. Global Opt.*, 11(4):341–359, 1997.

[16] J. Teo. Exploring dynamic self-adaptive populations in differential evolution. *Soft Comput.*, 10(8):673–686, 2006.

STOPPING CRITERIA FOR CONSTRAINED OPTIMIZATION WITH PARTICLE SWARMS

Karin Zielinski, Rainer Laur
Institute for Electromagnetic Theory and Microelectronics
University of Bremen, Germany
{zielinski,rlaur}@item.uni-bremen.de

Abstract Although different mechanisms can be used for the termination of an optimization run, only two of them are frequently used in the literature. However, both methods have disadvantages, particularly for the optimization of real-world problems. Because especially for practical applications it is important when an optimization algorithm is terminated as they usually contain computationally expensive objective functions, the performance of several stopping criteria that react adaptively to the state of an optimization run is evaluated for a Particle Swarm Optimization algorithm in this work. The examination is done on the basis of a constrained single-objective power allocation problem. Suggestions from former work concerning stopping criteria for unconstrained optimization are verified and comparisons with results for Differential Evolution are made.

Keywords: Constraints, Particle swarm optimization, Stopping criteria

1. Introduction

For theoretical aspects of evolutionary algorithms (or population-based search algorithms in general) stopping criteria are usually not important. However, for practical applications the choice of stopping criteria can significantly influence the duration of an optimization run. Due to different stopping criteria an optimization run might be terminated before the population has converged, or computational resources might be wasted because the optimization run is terminated late. Real-world problems mostly contain computationally expensive objective functions that may result in optimization runs that take several days, thus wasting of computational resources has to be prevented.

In the literature mostly two stopping criteria are applied: Either an error measure in dependence on the known optimum is used or the number of function evaluations is limited to fe_{max}. These criteria are perfectly suitable for e.g. comparing the performance of different algorithms but for solving real-world problems there are some drawbacks. The first mentioned method has the disadvantage that the optimum has to be known, so it is generally not applicable to real-world problems. The second method is highly dependent on the objective function. Because generally no correlation can be seen between an optimization problem and the required number of function evaluations, fe_{max} has to be determined by trial-and-error methods usually. Because the number of function evaluations that is needed for convergence is subject to fluctuations due to the randomness involved in the evolutionary process, a safety margin for fe_{max} is needed. The fluctuations can be significant as can be seen in [7] where a test suite of 24 functions has been examined and the standard deviation of function evaluations for reaching a predefined error measure was up to 180,000. If a real-world problem with an unknown optimum would result in a similar standard deviation, it would be difficult to choose fe_{max}.

Therefore, it would be better to use stopping criteria that consider knowledge from the state of the optimization run. Thus, the time of termination would be determined adaptively, so function evaluations could be saved.

Several stopping criteria are reviewed in [8] and [9] that are sensitive to the state of the optimization run by observing the improvement, movement or distribution of the population members. In [8] stopping criteria are tested for unconstrained single-objective optimization using Particle Swarm Optimization (PSO) [1], and Differential Evolution (DE) [3], while in [9] the criteria have been adapted for constrained single-objective problems using DE. In this work it will be examined if the suggestions regarding stopping criteria for PSO from [8] hold for the real-world problem of optimizing a power allocation scheme. Furthermore, a comparison with the results for DE in [9] will be done.

This work is organized as follows: Section 2 gives a brief introduction to Particle Swarm Optimization and in Section 3 the used stopping criteria are reviewed. In Section 4 results are shown and Section 5 closes with conclusions.

2. Particle Swarm Optimization

Particle Swarm Optimization is derived from the behavior of social groups like bird flocks or fish swarms. Optimization is achieved by giving each individual in the search space a memory for its previous successes, information about successes of a social group and providing a way to incorporate this knowledge into the movement of the individual. Therefore, each individual (called particle) is characterized by its position $\vec{x}_i$, its velocity $\vec{v}_i$, its personal best position $\vec{p}_i$ and its neighborhood best position $\vec{p}_g$. Several neighborhood topologies have been developed [4]. In this work the *von-Neumann* topology is used as it showed promising results in the literature, e.g., in [2].

The dynamic behavior of PSO is generated by the update equations for velocity and position of the particles:

$$\vec{v}_i(t+1) = w \cdot \vec{v}_i(t) + c_1 r_1[\vec{p}_i(t) - \vec{x}_i(t)] + c_2 r_2[\vec{p}_g(t) - \vec{x}_i(t)] \quad (1)$$

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1) \tag{2}$$

Due to these equations the particles are drawn towards their personal best position and their neighborhood best position, and furthermore the velocity of the previous iteration is kept weighted with the inertia weight w. Other parameters are c_1 and c_2 which influence the impact of the cognitive and social component, respectively. To add a stochastic element to the movement of the particles, the numbers r_1 and r_2 are chosen randomly from the interval [0,1] in each iteration. Further parameters of PSO are the population size NP and the maximum velocity V_{max}.

The parameter settings for this examination are derived from a parameter study using the same optimization problem (yet unpublished): $w = 0.6$, $c_1 = 0.4$, $c_2 = 1.4$, $NP = 64$, $V_{max} = \frac{1}{2}\left(X_{max} - X_{min}\right)$.

Constraint-handling is done by modifying the replacement procedure for personal and neighborhood best positions [5]. If two vectors $\vec{a}$ and $\vec{b}$ are compared, $\vec{a}$ is preferred if both vectors are feasible and $\vec{a}$ has a better objective function value, or if both are infeasible and $\vec{a}$ has the lower sum of constraint violation, or if $\vec{a}$ is feasible and $\vec{b}$ is not. No additional parameters are needed for this constraint-handling technique.

3. Stopping Criteria

Stopping criteria are needed to terminate the execution of optimization algorithms. In contrast to using a maximum number of function evaluations as a stopping condition, other criteria have the advantage of reacting adaptively to the state of the optimization run, thus function evaluations can be saved. Unfortunately, it seems to be impossible to define a stopping criterion without introducing one or more parameters. The parameter settings generally depend on the given optimization problem. However, it should be investigated if there are stopping criteria for which the parameter settings are robust to changes or if parameters can be set depending on certain aspects of the problem. It is assumed that the general behavior of different optimization problems to stopping criteria is similar. It should be kept in mind that limiting the number of function evaluations as a stopping criterion also incorporates the choice of a problem-dependent parameter. Therefore, it is favorable to examine other possibilities for stopping that contain the advantage of reacting adaptively to the state of the optimization run.

In the following the stopping criteria that incorporate information from the state of the optimization run are reviewed shortly. Note that there is a change compared to [8]: Instead of using the current positions $\vec{x}_i$ for the calculation of stopping conditions, the personal best positions $\vec{p}_i$ are used here.

Improvement-based criteria terminate an optimization run if only small improvement is made. Three different conditions are used here:

- *ImpBest*: The improvement of the best objective function value is monitored. If it falls below a given threshold t for a number of generations g, the optimization run is terminated.
- *ImpAv*: Similar to *ImpBest*, but instead of observing the best objective function value, the average value computed from the whole population is checked.
- *NoAcc*: It is observed if any new $\vec{p}_i$ are accepted in a specified number of generations g. For DE this criterion is slightly different (the acceptance of new population members is considered).

For *movement-based criteria* not the improvement but the movement of individuals is regarded. Two variants of movement-based criteria are considered that differ in the regarded space:

- *MovObj*: The movement of the individuals with respect to their objective function value (objective space) is examined if it is below a threshold t for a number of generations g. *MovObj* is different from *ImpAv* only if the regarded algorithm allows deterioration of the individuals' objective function value. This is the case for PSO in contrast to DE, but as $\vec{p}_i$ are considered here instead of $\vec{x}_i$, *MovObj* $=$ *ImpAv* holds in this case also. Therefore, this criterion is not regarded further in this work.
- *MovPar*: The movement with respect to positions (parameter space) is checked if it is below a threshold t for a number of generations g.

The *distribution-based criteria* consider the diversity in the population. If the diversity is low, the individuals are close to each other, so it is assumed that convergence has been obtained.

- *StdDev*: It is checked if the standard deviation of positions is below a threshold m.
- *MaxDist*: The distance from every population member to the best individual is observed. The optimization run is stopped if the maximum distance is below a threshold m.
- *MaxDistQuick*: *MaxDistQuick* is a generalization of *MaxDist*. Instead of using the whole population for the computation of the maximum distance to the best population member, only the best p% of the individuals are regarded. To achieve this, a quicksort algorithm is employed for sorting the individuals due to their objective function value.
- *Diff*: The difference between best and worst objective function is checked if it is below a threshold d. A further demand is that at least p% of the individuals are feasible because otherwise *Diff* could lead to undesired results if e.g. only two individuals are feasible and they are close to each other by chance. In contrast to the previous three criteria that are used in parameter space, *Diff* considers objective space.

Because functions have different features it may be beneficial to couple several criteria. Up to now two *combined criteria* have been regarded:

- *ComCrit*: This criterion is a combination of *ImpAv* and *MaxDist*. Only if the condition of *ImpAv* is fulfilled, *MaxDist* is checked.
- *Diff_MaxDistQuick*: *Diff* is a criterion that is rather easy to check, but it fails with flat surfaces. Therefore, if its condition has been fulfilled, the *MaxDistQuick* criterion is checked afterwards.

4. Results

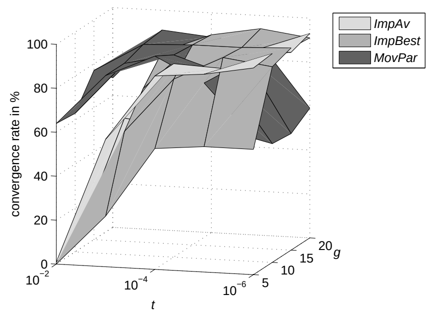
As a basis for the examination a real-world problem was used that consists of optimizing a power allocation scheme for a Code Division Multiple Access (CDMA) system [9]. The overall power is minimized considering the powers of 16 individual users as parameters. Because multiple users send data simultaneously in a CDMA system, multi-user interference degrades the system performance. By the application of a parallel interference cancelation technique the multi-user interference can be estimated and subtracted from the received signal before detection, thus the system performance improves. Therefore, the convergence of the parallel interference cancelation technique is incorporated in the optimization problem as a constraint.

In the following results are shown sorted according to the type of stopping criterion. Optimization runs are regarded as successful if an objective function value of $f(x) \leq 18.5$ has been reached [9]. As performance measures the convergence rate and the success performance (mean number of function evaluations weighed with the total number of runs divided by the number of successful runs) are given. To allow easy comparisons, figures showing success performances are scaled to 20,000. A maximum number of generations $G_{max} = 1000$ is used to terminate the algorithm if the examined stopping criteria do not lead to termination in appropriate time. If a run is not stopped before G_{max} is reached, the run is considered unsuccessful.

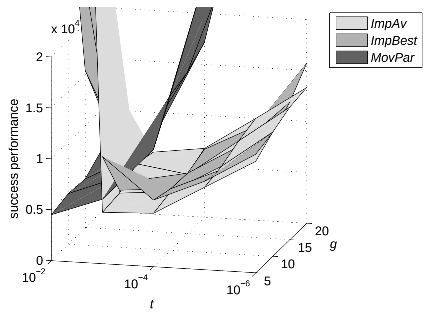
4.1 Improvement- and Movement-Based Criteria

Because *ImpAv*, *ImpBest* and *MovPar* rely on similar mechanisms, the convergence rate and success performance of these criteria are displayed together. Considering the convergence rate, almost no dependence on the number of generations g is observable (Figure 1(a)). For decreasing values of the improvement threshold t generally the convergence rate increases, except for *MovPar* that was not able to terminate several runs before reaching G_{max} for small settings of t.

The success performance of *ImpAv* and *MovPar* (Figure 1(b)) has similar characteristics as for DE in [9]. For *ImpBest* the results are different: The success performance for $g = 5$ is considerably better for PSO. Furthermore,



(a) Convergence rate



(b) Success performance

Figure 1. Results for criteria *ImpAv*, *ImpBest* and *MovPar*.

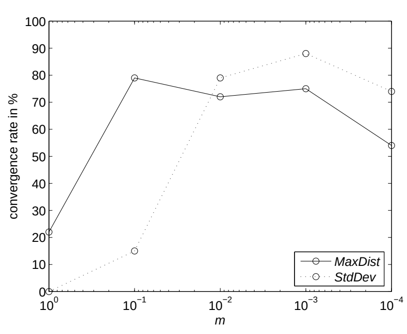
the success performance is dependent on t and independent from g whereas for DE it depends more on g than on t. The reason for the different results is not clear yet. It is interesting to note that although the convergence rate of *MovPar* is smaller for $t = 10^{-2}$ than for $t = 10^{-4}$, the success performance is better due to a large difference in the average number of function evaluations.

The results for *ImpAv* and *ImpBest* are considerably better here than in [8] for unconstrained single-objective problems. For *ImpAv* the reason might be that the personal bests are regarded here instead of the current positions, but *ImpBest* did not change because only the global best result is regarded. In contrast, for *MovPar* the results are worse. However, suitable parameter settings for *ImpAv* and *ImpBest* cannot be derived from knowledge about the optimization problem. Furthermore, it is indicated in [8] that problems arise for functions with a flat surface, but it is usually not known in advance if a function possesses this property. Therefore, it will be necessary to do examinations on parameter settings for the application of these stopping criteria. Based on the examined problem parameter settings of $g \approx 10 \ldots 15$ and $t \approx 10^{-5} \ldots 10^{-4}$ are recommended. However, these settings are dependent on the optimization problem and the desired accuracy.

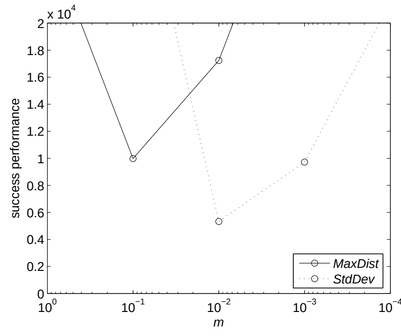
Criterion *NoAcc* showed good results for DE in [9] but not a single run could be terminated before reaching G_{max} for PSO. Apparently, the personal best positions improve too often to allow a stopping criterion like *NoAcc*.

4.2 Distribution-Based Criteria

For *MaxDist* the convergence rate does not get above 80 % because of runs that could not be terminated before reaching G_{max}. The results for *StdDev* are shifted in contrast to *MaxDist* and higher convergence rates are reached (Figure 2(a)). Furthermore, *StdDev* yields a lower minimum success performance

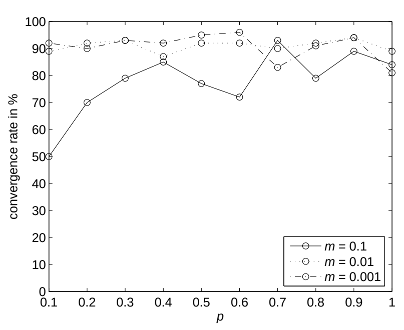


(a) Convergence rate

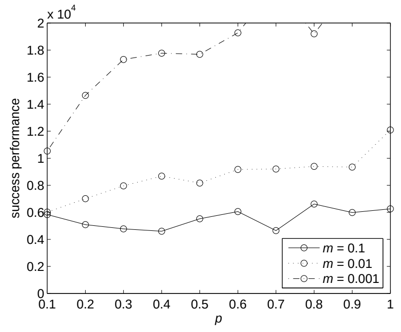


(b) Success performance

Figure 2. Results for criteria *MaxDist* and *StdDev*.



(a) Convergence rate



(b) Success performance

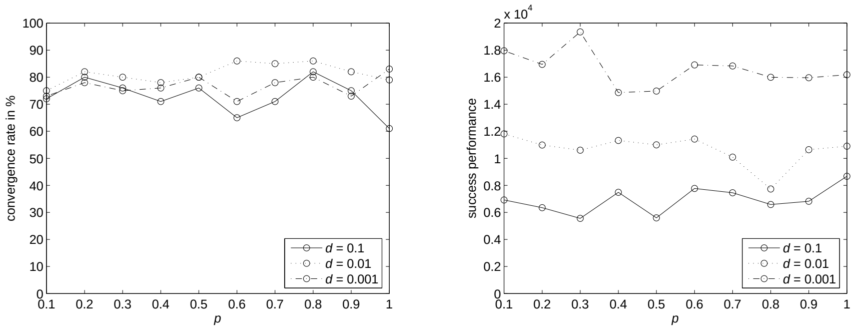
Figure 3. Results for criterion *MaxDistQuick*.

than *MaxDist* (Figure 2(b)). The performance is highly dependent on the setting of m. However, it is connected to the desired accuracy. Similar results have been found in [9] for DE. Compared to DE, the same settings of parameter m yield the lowest success performances for *MaxDist* and *StdDev*, respectively.

The convergence rate and success performance of *MaxDistQuick* is given for $10^{-3} \leq m \leq 10^{-1}$ in Figures 3(a) and 3(b). Other parameter settings are omitted because the success performance was above 20,000. The convergence rate is fluctuating for $m = 0.1$ with different settings of p, indicating that the performance is not robust for these parameter settings. For $m = \{10^{-2}, 10^{-3}\}$ and varying p the convergence rate is approximately constant but the success performance rises with increasing p. Thus, a similar result is obtained as in [8]: Because less function evaluations are needed for convergence if smaller values of p are used and the convergence probability is not compromised, it is recommended to use e.g. $0.3 \leq p \leq 0.5$. For DE the success performance depends less on p and increases more strongly with decreasing m. In spite of the increased computational effort for the incorporated quicksort algorithm, *MaxDistQuick* is considered to be superior to *MaxDist* and *StdDev* for PSO. For future work it would be also interesting to evaluate a similar criterion based on standard deviation instead of maximum distance.

It may be confusing that the success performance for *MaxDistQuick* with $p = 1$ is not equal to the results of *MaxDist*. The reason is that the success performance is sensitive to even small changes in the number of successful runs. If the average number of function evaluations is regarded, the results from *MaxDistQuick* with $p = 1$ and *MaxDist* are similar (not shown here).

For criterion *Diff* no definite trend can be observed regarding the demanded percentage p of feasible individuals in the population (Figures 4(a) and 4(b)) which is assumed to be due to the fact that all individuals get feasible quite fast here. Similar results were found for DE in [9]. However, the success performance depends on the difference threshold d as expected. As for the other distribution-based criteria, the setting of d is dependent on the desired accuracy. The highest convergence rate is achieved with $d = 10^{-2}$ but although $d = 10^{-1}$ results in a worse convergence rate, the success performance is better.



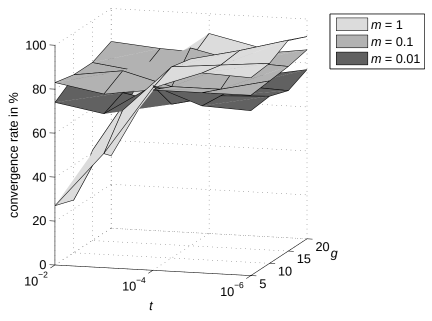
(a) Convergence rate (b) Success performance

Figure 4. Results for criterion *Diff*.

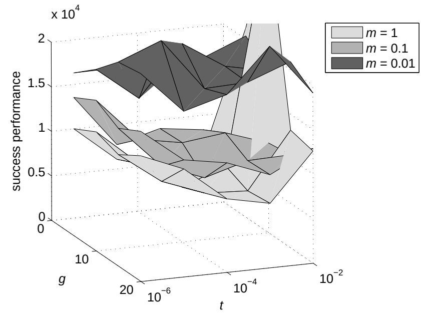
Criterion *Diff* is advantageous in contrast to the distribution-based criteria in parameter space if several parameter combinations yield the same objective function value. However, it is likely to produce bad results for a function with a flat surface.

4.3 Combined Criteria

The convergence rate and success performance for both combined criteria are given for $m \geq 10^{-2}$ because smaller values of m lead to success performances larger than 20,000 (Figures 5(a), 5(b), 6(a), and 6(b)). The results are different than for DE as the success performance increases less with decreasing value of m. Especially for *Diff_MaxDistQuick* the results are rather independent from m. However, a strong dependence on d can be seen, in particular for the success performance. For the combined criteria the dependence of parameter settings on the desired accuracy of the results cannot be seen anymore, so in general it might be easier to use the individual criteria.

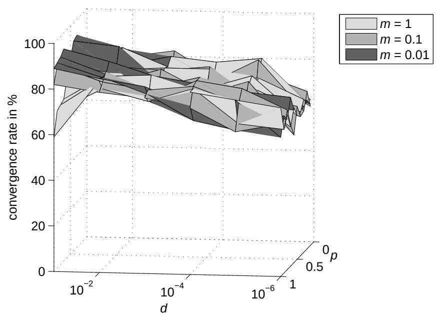


(a) Convergence rate

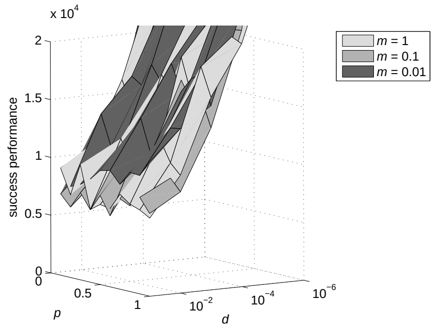


(b) Success performance

Figure 5. Results for criterion *ComCrit*.



(a) Convergence rate



(b) Success performance

Figure 6. Results for criterion *Diff_MaxDistQuick*.

5. Conclusions

In this work stopping criteria were studied that react adaptively to the state of an optimization run based on improvement, movement or the distribution of individuals. In contrast to other examinations, not the current positions but the personal best positions were used for the calculations. It was shown that the stopping criteria can be used for constrained problems using PSO. A similar behavior as for DE could be found for several stopping criteria. It would be interesting to make comparisons with other Evolutionary Algorithms in future work.

Although parameter settings have to be determined in dependence on the used optimization problem, general statements could be made. It was not possible to determine one criterion that will be best for all problems, but because of their adaptive nature generally improved performance for real-world problems is expected in contrast to termination after a limited number of function evaluations.

For multi-objective optimization it will be even more challenging to define suitable stopping criteria [6] but also even more important because usually the population will not converge to one point in the search space but to the Pareto-optimal front, thus using error measures is difficult. Therefore, it is an interesting field of research for future work.

References

[1] J. Kennedy and R.C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, 2001.

[2] J. Kennedy and R. Mendes. Population Structure and Particle Swarm Performance. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2002)*, pages 1671–1676, Honolulu, HI, USA, 2002.

[3] J. Lampinen and R. Storn. Differential Evolution. In G.C. Onwubolu and B. Babu, editors, *New Optimization Techniques in Engineering*, pages 123–166. Springer-Verlag, Berlin Heidelberg, 2004.

[4] R. Mendes, J. Kennedy, and J. Neves. The Fully Informed Particle Swarm: Simpler, Maybe Better. *IEEE Trans. Evol. Comput.*, 8(3):204–210, 2004.

[5] G. T. Pulido and C. A. Coello Coello. A Constraint-Handling Mechanism for Particle Swarm Optimization. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2004)*, volume 2, pages 1396–1403, Portland, OR, USA, 2004.

[6] O. Rudenko and M. Schoenauer. A Steady Performance Stopping Criterion for Pareto-based Evolutionary Algorithms. In *Proc. 6th International Multi-Objective Programming and Goal Programming Conference*, Hammamet, Tunisia, 2004.

[7] K. Zielinski and R. Laur. Constrained Single-Objective Optimization Using Particle Swarm Optimization. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2006)*, pages 1550–1557, Vancouver, BC, Canada, 2006.

[8] K. Zielinski, D. Peters, and R. Laur. Stopping Criteria for Single-Objective Optimization. In *Proc. 3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2005)*, Singapore, 2005.

[9] K. Zielinski, P. Weitkemper, R. Laur, and K.-D. Kammeyer. Examination of Stopping Criteria for Differential Evolution based on a Power Allocation Problem. In *Proc. 10th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM'06)*, Braşov, Romania, 2006.

NON-PARAMETRIC GENETIC ALGORITHM

Gregor Papa
Computer Systems Department
Jožef Stefan Institute, Ljubljana, Slovenia
gregor.papa@ijs.si

Abstract In this paper the non-parametric genetic algorithm is presented. It does not need any predefined operator control parameters value as population size, number of generations, probabilities of crossover and mutation are. Suitability and efficiency of the proposed algorithm were evaluated by the CEC 2006 benchmark functions. The results show the lack of suitability of non-parametric genetic algorithm when dealing with optimization problems with many unfeasible zones. Even though the non-parametric genetic algorithm is very fast, it still needs some improvements.

Keywords: Genetic algorithm, Non-parametric, Self adaptation

1. Introduction

The aim of many researchers and developers of heuristic optimization algorithms is to make an algorithm that would be able to solve the given problem without any human intervention for setting the suitable control parameters [1, 3, 4]. Among different optimization techniques genetic algorithm (GA) is popular due to its simplicity, but there are very important parameters that need to be set in advance to ensure effective optimization. In this paper the non-parametric genetic algorithm (NPGA) is described [5]. This algorithm does not need any control parameter, e.g., population size, number of generations, probabilities of crossover and mutation, to be set in advance, but it sets them according to complexity of the problem and according to convergence of the solution.

The suitability and efficiency of the proposed algorithm were evaluated by the CEC 2006 benchmark functions [2].

In the second section the NPGA and its operators are described in details; in the third section the test functions are presented; while the fourth and the fifth section present the results of the evaluation and conclusion, respectively.

```
    Genetic Algorithm {
      setting the initial population $S$ of random individuals;
      while stopping criterion not met {
        evaluate each individual;
        select two parents;
        create two offspring by crossing the parents;
        mutate the offspring }
    }
```

Figure 1. Outline of the genetic algorithm.

2. Non-Parametric GA

The main advantage of NPGA over the basic GA is the fact that NPGA can set the variable control parameters like population size, number of generations, probabilities of crossover and mutation by itself in the initialization phase and during the optimization process. The values of those parameters depend on the complexity of the problem that needs to be solved and according to the behavior and convergence of the found solutions. The pseudo code of the algorithm is presented in Figure 1.

There is no difference between the general genetic algorithm and non-parametric genetic algorithm when comparing their pseudo codes. The main difference is in the way how those genetic operators (selection, crossover and mutation) are executed, what are their control parameters values, how are those parameters determined, and how do they vary.

In the following subsections the behavior of the genetic operators and the way for determining the parameters that control these operators are presented.

2.1 Setup

The chromosome that represents the problem to be solved is constructed upon the number of the variables of the problem and their dependencies. For the n independent variables the chromosome would look like the string of n values in the order as described in the input specification of the problem, while for n dependent variables the n positions and the order in the chromosome would represent dependencies as described in the input specification of the problem. In the second case the interdependent variables would be placed together or closer in the chromosome.

The values in the initial chromosome are set to random value in the range between the maximum and minimum value for each variable.

```
**Stopping criteria** {
    $NoChg$ = number of generations without improvement;
    $CurGen$ = current generation number;
    $VarNum$ = number of variables;
    **if** $(NoChg \leq \frac{1}{3}\, CurGen \wedge CurGen < 3\, PopSize) \vee$
            $(NoChg \leq \frac{1}{5}\, CurGen \wedge CurGen \geq 3\, PopSize)$ **then**
        continue optimization;
    **else**
        stop optimization
}
```

Figure 2. Number of generations – stopping criteria.

2.2 Initialization

If the chromosome that represents the problem and its complexity is large, than also the population size is larger. This is needed to ensure higher versatility among the chromosomes in the population. Therefore more solutions can be searched in parallel in each iteration. The population size is proportional to chromosome size, i.e., problem complexity.

In the NPGA the population size, *PopSize*, depends on the number of variables (*VarNum*) and the ranges of all variables to be optimized. See Eqn. (1) for details.

$$PopSize = 3VarNum + \ln(100VarNum) + \ln(Range), \tag{1}$$

where

$$Range = \sum_{j=1}^{VarNum} \big((variable_{j_{max}} - variable_{j_{min}})(VarRes_j + 1)\big) \tag{2}$$

and *VarRes* represents the resolution, i.e., number of decimal places, of the given variable.

The number of generations depends on the convergence speed of the best solution found. Optimization is running while better solution is found every few generations. But when there is no improvement of the best solution for a couple of generations, the optimization process stops.

In the NPGA the optimization process stops when there is no improvement of the best solution for one third of the past generations when the number of generations is smaller than three-times of the *PopSize*; or if there is no improvement for one fifth of the past generations after the number of generations is larger than three-times of the *PopSize*. See Figure 2 for details.

```
Mutation – above average {
    NoChg = number of generations without improvement;
    CurGen = current generation number;
    shift = NoChg / CurGen;
    if $s_i > s_{best}$ then {
        randomly choose variable $j$ of the solution $i$;
        if $s_{i_j} < s_{best_j}(\frac{1}{2} + shift) \vee s_{i_j} > s_{best_j}(\frac{3}{2} - shift)$ then
            mutate the variable $j$ }
}
```

Figure 3. Choosing the variable for mutation in above average chromosome.

2.3 Crossover

The crossover takes place in each generation. There is $\frac{1}{2}PopSize$ mates and two crossover points on the chromosomes are randomly selected for each pair. After the exchange of values of the mated chromosomes on places between the two crossover points two new offspring are created.

Among those four candidates (two parents and two offspring) only two are passed to the next generation. The first one is one of the offspring, which is randomly chosen, and the second one is one of the all four candidates.

This procedure ensures that more offspring are passed to the next generations, but also some parents have chances to proceed to the next generation.

2.4 Mutation

Every chromosome is the subject of mutation. If the fitness s_i of the chromosome is above the average in the current population (the subject of the optimization is minimization) then the randomly chosen position s_{i_j} in the chromosome is mutated if the value of the position is smaller than the shifted value of the best solution s_{best} or if the value is larger than the shifted value of the best solution. There is always only the small number of mutated positions in each chromosome – this number depends on the number of generations when there was no improvement of the best solution. The larger is the number of generations without improvements of the best solution, the larger is the number of mutated positions. The code is presented in Figure 3.

If the fitness of the chromosome is below the average in the current population then the randomly chosen position s_{i_j} in the chromosome is mutated if the value of the position is larger than the shifted value of the best solution s_{best} and if the value is smaller than the shifted value of the best solution. Again, only the small number of positions is mutated in each chromosome. The code for below average solutions is presented in Figure 4.

```
Mutation – below average {
   NoChg = number of generations without improvement;
   CurGen = current generation number;
   shift = NoChg / CurGen;
   if $s_i \leq s_{best}$ then {
      randomly choose variable $j$ of the solution $i$;
      if $s_{i_j} > s_{best_j}(\frac{3}{4} + \frac{1}{2}\,shift) \wedge s_{i_j} < s_{best_j}(\frac{5}{4} - \frac{1}{2}\,shift)$ then
         mutate the variable $j$ }
}
```

Figure 4. Choosing the variable for mutation in below average chromosome.

```
Mutation – moves {
   NoChg = number of generations without improvement;
   VarNum = number of variables;
   resolution = smallest change of $variable_j$;
   range = $variable_{j_{max}} - variable_{j_{min}}$;
   if NoChg < VarNum then {
      range = (0.05 range − resolution)/ VarNum;
      move = (VarNum − NoChg) range }
   else if NoChg ≥ VarNum ∧ NoChg < 2 VarNum then
      move = resolution;
   else if NoChg ≥ 2 VarNum ∧ NoChg < 3 VarNum then {
      range = (0.05 range − resolution)/ VarNum;
      move = (NoChg −2 VarNum)· range }
   else {
      range = 0.05 range / VarNum;
      move = (NoChg −3 VarNum)· range }
   randomly chose the direction as 1 or -1;
   $variable_j = variable_j + direction \cdot move$
}
```

Figure 5. Performing the moves in mutation operator.

In each case described above the volume of the change of the variable value is calculated by the procedure described in Figure 5.

3. Test Functions

The experiments for the evaluation of the NPGA were performed by CEC 2006 benchmark functions defined for constrained real-parameter optimization. The set of test functions consists of 24 functions. These functions are particulary useful for testing the algorithm that tries to solve problems in which the optimum lies in the boundary between the feasible and the infeasible regions or when the feasible region is disjoint.

For the purpose of testing the NPGA algorithm, only the first six functions were used. Among them there are polynomial, nonlinear, quadratic, and cubic functions. The details of the used test functions are presented in Table 1.

Table 1. CEC 2006 functions properties

Function	Number of variables	Type of function	Feasible region	Optimal value
$g01$	13	quadratic	0.0111 %	-15.00000000
$g02$	20	nonlinear	99.9971 %	-0.8036191
$g03$	10	polynomial	0.0000 %	-1.0005001
$g04$	5	quadratic	52.1230 %	-30665.53867
$g05$	4	cubic	0.0000 %	5126.49671400
$g06$	2	cubic	0.0066 %	-6961.8138755

4. Results

The NPGA run 30-times over each test function. The experiments were done on 2 GHz computer, and each run took approximately 1–2 minutes (depends on function). However time complexity was not the subject of this evaluation.

The best, worst and average value of the found solutions after 30 runs are presented in Table 2. All values are optimized with the precision of 6 decimal places.

Table 2. Results of NPGA testing with CEC 2006 functions

	$g01$	$g02$	$g03$	$g04$	$g05$	$g06$
Best	-15.577020	-0.767520	–	-32,018.405580	5,204.730340	-7,865.806500
Average	-13.854983	-0.697037	–	-31,855.884334	5,334.592292	-7,471.461654
Worst	-11.842490	-0.584740	–	-31,244.373170	5,454.589250	-6,836.326270
Avrg FES	225,436	307,088	–	34,020	3,546	1,199

Since the crossover and mutation are controlled by the algorithm itself, the so called virtual settings for those operators are presented in Table 3. The values are virtual, since they were calculated a posteriori upon the information obtained by the algorithm about how many chromosomes were actually mated for crossover and how many positions in the chromosomes were actually mutated.

Figure 6 represents the fitness value convergence in case of test function $g01$. It is represented as average over all 30 runs.

The results presented in Table 2 show that NPGA is able to come close to the optimal solution very quickly, even it the optimal solution is surrounded by

Table 3. Virtual values of control parameters

	$g01$	$g02$	$g03$	$g04$	$g05$	$g06$
POPULATION SIZE	53	74	40	27	27	18
NUMBER OF GENERATIONS	2848	4234	11256	767	97	25
PROBABILITY OF CROSSOVER	0.668	0.667	0.667	0.665	0.667	0.644
PROBABILITY OF MUTATION	0.161	0.146	0.031	0.029	0.380	0.168

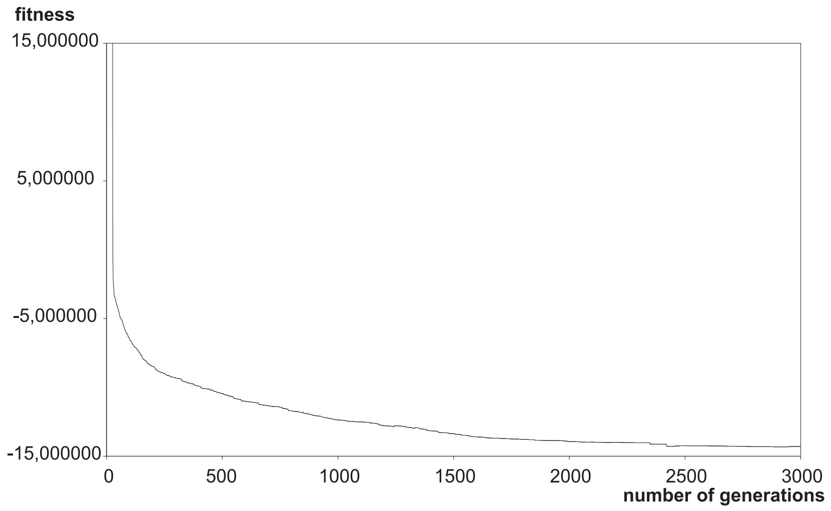


Figure 6. Fitness convergence for function $g01$.

unfeasible regions. In most examples the algorithm was able to come out of the unfeasible regions, except for function $g03$, where the algorithm was unable to come to the optimal solution. To improve the performance of the algorithm, few more changes inside the algorithm, to calculate the control parameters, need to be done.

However, regarding the numbers presented in Table 3 NPGA behaves similar to some parameter-needed GAs, since the virtual values of control parameters (probabilities of crossover and mutation) are in the order of magnitude as those reported in the literature for this kind of test functions.

5. Conclusion

In this paper the non-parametric genetic algorithm is presented. This algorithm does not need any predefined operator control parameter values as population size, number of generations, probabilities of crossover an mutation are. Suitability and efficiency of the proposed algorithm were evaluated by the CEC 2006 benchmark functions. The results show the lack of suitability of non-parametric genetic algorithm when dealing with optimization problems with many unfeasible zones. Even though the non-parametric genetic algorithm was much faster than the other algorithms, it still needs some improvements.

References

[1] J. Brest, V. Žumer, and M. Sepesy Maučec. Self-adaptive Differential Evolution Algorithm in Constrained Real-Parameter Optimization. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2006)*, pages 215–222, Vancouver, BC, Canada, 2006.

[2] J.J. Liang, T.P. Runarsson, E. Mezura-Montes, M. Clerc, P.N. Suganthan, C.A. Coello Coello, and K. Deb. Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization. Technical Report #2006005, Nanyang Technological University, Singapore, March, 2006. `http://www.ntu.edu.sg/home/EPNSugan`.

[3] S. Liu, M. Mernik, and B.R. Bryant. Parameter Control in Evolutionary Algorithms by Domain-Specific Scripting Language PPCEA. In *Proc. International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004)*, pages 41–50, Ljubljana, Slovenia, 2004.

[4] F.G. Lobo. The Parameter-Less Genetic Algorithm: Rational And Automated Parameter Selection For Simplified Genetic Algorithm Operation. PhD thesis, Universidade Nova de Lisboa, Lisboa, 2000.

[5] G. Papa. Concurrent operation scheduling and unit allocation with an evolutionary technique in the process of integrated-circuit design. PhD thesis, University of Ljubljana, Ljubljana, 2002.

TAKEOVER TIME IN PARALLEL POPULATIONS WITH MIGRATION

Günter Rudolph
Department of Computer Science
University of Dortmund, Germany
guenter.rudolph@uni-dortmund.de

Abstract The term *takeover time* regarding selection methods used in evolutionary algorithms denotes the (expected) number of iterations of the selection method until the entire population consists of copies of the best individual, provided that the initial population consists of a single copy of the best individual whereas the remaining individuals are worse. Here, this notion is extended to parallel subpopulations that exchange individuals according to some migration paths modelled by a directed graph. We develop upper bounds for migrations on uni- and bidirectional rings as well as arbitrary connected graphs where each vertex is reachable from every other vertex.

Keywords: Takeover time, Spatially structured population, Migration model

1. Introduction

The term *takeover time* regarding selection methods used in evolutionary algorithms (EAs) was introduced by Goldberg and Deb [7]. Suppose that a finite population of size n consists of a single best individual and $n - 1$ worse individuals. The takeover time of some selection method is the (expected) number of iterations of the selection method until the entire population consists of copies of the best individual.

The calculations in [7] for spatially unstructured (i.e., panmictic) populations implicitly assume that at least one copy of the best individual is kept in the population although some selection method may erase all best copies by chance. If a selection method is *elitist*, i.e., the best individual survives selection with probability 1, this kind of extinction is precluded. At a first glance it is surprising that most results on the takeover time are approximations (without bounds) [7] or obtained numerically by an underlying Markov chain model [2, 11].

Apparently, selection in panmictic populations is the most difficult case for deriving rigorous results on the takeover time. If only a single individual is generated in each generation (steady-state EA) the Markov model looses some of its complexity as has been shown by Smith and Vavak [11]. Mathematically rigorous results have been provided by Rudolph [9, 10] for some of these non-generational selection methods. In case of populations with a spatial structure (at the level of individuals) the notion of the takeover time must be extended appropriately. This has been done by Rudolph [8] who developed bounds on the takeover time for arbitrary connected population structures and even an exact expression for a structure like a ring. These results have been extended by Giacobini et al. [4, 5, 6].

Recently, Alba and Luque [1] have considered spatially structured populations that are structured at the level of subpopulations (in contrast to individuals). In this population model the subpopulations are panmictic and from time to time some individuals migrate between the subpopulations according to some connectivity graph: The vertices of the graph are the subpopulations whereas the directed edges are the migration paths. In [1] the authors develop a plausible approximation (without bounds) for some special cases.

This was the starting point of this work: We show how to derive rigorous bounds for the takeover time for parallel populations with migration. For this purpose some mathematical facts are introduced in Section 2 before the analysis is presented in Section 3.

2. Mathematical Preliminaries

In the course of the analysis given in Section 3 we need bounds on Harmonic numbers:

Definition 1
The symbol H_n denotes *nth Harmonic number* for some $n \in \mathbb{N}$ where

$$H_n = \sum_{i=1}^{n} \frac{1}{i} \,.$$

Likewise, the *nth Harmonic number of 2nd order $H_n^{(2)}$* is given by

$$H_n^{(2)} = \sum_{i=1}^{n} \frac{1}{i^2}$$

for $n \geq 1$. □

Notice that

$$\log(n) \leq H_n \leq \log(n) + 1$$

for $n \geq 2$ and

$$1 \leq H_n^{(2)} \leq \frac{\pi^2}{6}$$

for $n \geq 1$.

Definition 2
A random variable G is *geometrically distributed* with support $\mathbb{N}$ if $\mathsf{P}\{ G = k \} = p\,(1-p)^{k-1}$ for some $p \in (0,1) \subset \mathbb{R}$. □

The expectation and variance of G are

$$\mathsf{E}[\,G\,] \;=\; \frac{1}{p} \text{ resp. } \mathsf{V}[\,G\,] \;=\; \frac{p}{(1-p)^2}. \tag{1}$$

Definition 3
Let $X_1, X_2, \ldots, X_n$ be independent and identically distributed (i.i.d.) random variables. Then $X_{1:n}$ denotes the minimum and $X_{n:n}$ the maximum of these random variables. □

Let $\mathsf{D}[\,X\,] = \sqrt{\mathsf{V}[\,X\,]}$ denote the standard deviation of some random variable X. There exists a general result regarding bounds on the expectation of the minimum and maximum:

Theorem 1 (David 1980, p. 59 and 63)
Let $X_1, X_2, \ldots, X_n$ be an i.i.d. sequence of random variables. The bounds

$$\begin{aligned} \mathsf{E}[\,X_{1:n}\,] &\;\geq\; \mathsf{E}[\,X_1\,] - \frac{n-1}{\sqrt{2\,n-1}}\,\mathsf{D}[\,X_1\,] \\ \mathsf{E}[\,X_{n:n}\,] &\;\leq\; \mathsf{E}[\,X_1\,] + \frac{n-1}{\sqrt{2\,n-1}}\,\mathsf{D}[\,X_1\,] \end{aligned}$$

are valid regardless of the distribution of the X_i. □

3. Analysis

Let $\mathcal{G} = (V, E)$ denote a directed graph where each vertex $v \in V$ represents a subpopulation and each directed edge $e = (v, v') \in E$ a migration path from subpopulation v to subpopulation v'. Random variable $X_v^{(t)}$ specifies the number of individuals with best fitness at iteration $t \geq 0$ of subpopulation $v \in V$ with $X_k^{(0)} = 1$ for a single subpopulation k and $X_v^{(0)} = 0$ for $v \neq k$. The number of individuals s in each subpopulation is constant over time, identical for all subpopulations, and finite. Moreover, we make the following *general assumptions:*

(A1) Selection in subpopulations is elitist.

(A2) Migration takes place every mth generation with finite $m \in \mathbb{N}$.

(A3) Emmigration policy: a copy of the best individual travels along each migration path.

(A4) Immigration policy: replace the worst individual of the subpopulation with the immigrant (if it is better than the worst one).

Let $T_v = \min\{t \geq 0 : X_v^{(t)} = s\}$ be the random takeover time of subpopulation $v \in V$ and A_v the random arrival time, i.e., the number of iterations until the first individual with best fitness arrives at subpopulation $v \in V$. In general, the arrival times are not identically distributed. Their distributions depend on the connectivity or migration graph and in which subpopulation the initial best individual has emerged. If the migration path is vertex-symmetric (like Cayley graphs) the latter dependency vanishes. Here, we shall assume that the initial best individual emerges at vertex $v = 0$ and we rename the other vertices accordingly. Then

$$T = \max\{T_0, A_1 + T_1, A_2 + T_2, \ldots, A_n + T_n\} \tag{2}$$

is the *takeover time* of the migration model with $n + 1$ subpopulations considered here. Notice that random variables T_v are i.i.d. for $v \geq 1$ whereas the distribution of T_0 is different: Once a best copy has arrived at subpopulation $v \geq 1$, every mth generation at least one another best copy immigrates to this subpopulation regardless of the selection process within the subpopulation. Therefore it takes at most $m\,s$ iterations until all individuals in some subpopulation $v \geq 1$ are copies of the best individual regardless of the selection process. Thus,

$$T_v \leq m\,s \tag{3}$$

with probability 1 (w.p. 1) for $v \geq 1$. If m is large the bound above becomes useless since it is likely that the takeover event happens before the first migration interval is over. Therefore we define random variable T'_v which is the takeover time of subpopulation v if no further migration takes place once a best copy has arrived. As a consequence, we have

$$T_v \leq T'_v \tag{4}$$

w.p. 1 for all $v \geq 0$. Notice that $T'_0, T'_1, \ldots, T'_n$ are i.i.d. random variables.

3.1 Uni-Directional Ring Topology

Suppose that the subpopulations are placed at the vertices of a uni-directional ring. Then the takeover time in Eqn. (2) specializes to

$$T = \max\{T_0, m + T_1, 2\,m + T_2, \ldots, n\,m + T_n\} \tag{5}$$

for a finite migration interval $m \in \mathbb{N}$. Once a best individual has emerged at vertex 0 it takes m generations until this best individual migrates to vertex 1.

Now it takes again m iterations until a best copy migrates to vertex 2 and so forth. As soon as a best copy has arrived at some vertex v it takes T_v iterations at vertex v until all individuals are copies of the best individual. Evidently, T can be bracketed as follows:

$$n\,m + \min\{T_0, \ldots, T_n\} \le T \le n\,m + \max\{T_0, \ldots, T_n\}\,. \tag{6}$$

Using Eqn. (4) in the right hand side (r.h.s.) of inequality (6) we obtain the bound

$$T \le n\,m + \max\{T'_0, \ldots, T'_n\}$$

for the takeover time T and hence the bound

$$\mathsf{E}[\,T\,] \le n\,m + \mathsf{E}[\,T'_{n+1:n+1}\,] \tag{7}$$

for the expected takeover time. Usage of Eqn. (3) in the r.h.s. of inequality (6) yields $\mathsf{E}[\,T\,] \le n\,m + m\,s$ which leads to the bound

$$\mathsf{E}[\,T\,] \le n\,m + \min\{m\,s, \mathsf{E}[\,T'_{n+1:n+1}\,]\} \tag{8}$$

in consideration of Eqn. (7). Owing to Theorem 1 the bound in Eqn. (7) can be expressed in terms of the expectation $\mathsf{E}[\,T'_0\,]$ and standard deviation $\mathsf{D}[\,T'_0\,]$ of T'_0. We obtain

$$\mathsf{E}[\,T\,] \le n\,m + \mathsf{E}[\,T'_0\,] + \frac{n\,\mathsf{D}[\,T'_0\,]}{\sqrt{2\,n+1}}\,. \tag{9}$$

But as long as nothing is known about the selection operation within the subpopulations the distribution and therefore the moments of T'_0 remain unknown. Therefore we assume that each subpopulation runs a steady-state EA with a selection method that does not erase any copy of the best individual contained in the current population. In this case expectation and variance can be calculated as follows [9]: If i denotes the number of best copies of the current population then the value of i is a nondecreasing sequence. Let $p_{i,i+1}$ be the probability that the next population will contain $i+1$ best copies and $p_{i,i} = 1 - p_{i,i+1}$ the probability that the number of best copies will not change, provided the current number of best copies is i. Then the random number G_i of generations until i changes to $i+1$ is geometrically distributed with expectation and variance

$$\mathsf{E}[\,G_i\,] \;=\; \frac{1}{p_{i,i+1}} \;\text{ resp. }\; \mathsf{V}[\,G_i\,] \;=\; \frac{1-p_{i,i+1}}{p^2_{i,i+1}}$$

for $i = 1, \ldots, s-1$. Since $G_1, \ldots, G_{s-1}$ are mutually independent we obtain

$$\mathsf{E}[\,T'_0\,] \;=\; \sum_{i=1}^{s-1} \mathsf{E}[\,G_i\,] \;=\; \sum_{i=1}^{s-1} \frac{1}{p_{i,i+1}} \tag{10}$$

$$\mathsf{V}[\,T'_0\,] \;=\; \sum_{i=1}^{s-1} \mathsf{V}[\,G_i\,] \;=\; \sum_{i=1}^{s-1} \frac{1-p_{i,i+1}}{p^2_{i,i+1}} \tag{11}$$

for the takeover time T_0'. Next, we choose a specific selection method to exemplify our approach developed so far. The method called 'Replace Worst'-selection first draws two individuals at random with uniform probability. Subsequently the better one of the pair replaces the worst individual of the entire population. Therefore, i is incremented if at least one copy of the best individual is drawn. We obtain

$$p_{i,i+1} = 1 - \left(1 - \frac{i}{s}\right)^2 = \frac{i\,(2\,s - i)}{s^2}$$

and finally owing to Eqn. (10)

$$\mathsf{E}[\,T_0'\,] = \frac{1}{2}\,(s\,H_{2\,s-1} - 1). \tag{12}$$

The result for the expectation above can be found in [9] already. Here, we also need a result for the variance. According to Eqn. (11) we obtain

$$\begin{aligned}
\mathsf{V}[\,T_0'\,] &= \sum_{i=1}^{s-1} \frac{1 - p_{i,i+1}}{p_{i,i+1}^2} = \sum_{i=1}^{s-1} \frac{(s-i)^2}{i^2} \cdot \frac{s^2}{(2\,s-i)^2} \\
&\le \sum_{i=1}^{s-1} \left(\frac{s}{i} - 1\right)^2 \quad \text{since } \frac{s}{2\,s-i} \le 1 \\
&= \sum_{i=1}^{s-1} \left(\frac{s^2}{i^2} - \frac{2\,s}{i} + 1\right) \\
&= s^2\,H_{s-1}^{(2)} - 2\,s\,H_{s-1} + s - 1 \\
&\le s^2\,\frac{\pi^2}{6} - 2\,s\log(s-1) + s - 1 \quad \text{if } s \ge 3
\end{aligned}$$

and since $s/(2\,s-i) \ge 1/2$

$$\begin{aligned}
\mathsf{V}[\,T_0'\,] &= \sum_{i=1}^{s-1} \frac{(s-i)^2}{i^2} \cdot \frac{s^2}{(2\,s-i)^2} \ge \frac{1}{4} \sum_{i=1}^{s-1} \left(\frac{s}{i} - 1\right)^2 \\
&= \frac{1}{4}\,(s^2\,H_{s-1}^{(2)} - 2\,s\,H_{s-1} + s - 1) \\
&\ge \frac{1}{4}\,(s^2 - 2\,s\log(s-1) - s - 1\,)
\end{aligned}$$

revealing that $\mathsf{V}[\,T_0'\,] = \Theta(s^2)$ or $\mathsf{D}[\,T_0'\,] = \Theta(s)$. Insertion in Eqn. (9) yields the bound

$$\begin{aligned}
\mathsf{E}[\,T\,] &\le nm + \frac{s\,H_{2s-1} - 1}{2} + \frac{n}{\sqrt{2n+1}}\sqrt{\frac{s^2\,\pi^2}{6} - 2\,s\,\log(s-1) + s - 1} \\
&\le nm + \frac{s\,\log(2s)}{2} + \sqrt{\frac{n}{2}} \cdot \sqrt{\frac{s^2\pi^2}{6} + 1} \quad (\text{for } s \ge 2) \\
&= nm + \frac{s\,\log(2s)}{2} + s\,\pi\sqrt{\frac{n}{12}} \cdot \sqrt{1 + \frac{6}{s^2\pi^2}} \\
&\le nm + \frac{s\,\log(2s)}{2} + s\,\pi\sqrt{\frac{n}{6}} \\
&= \mathcal{O}(nm + s\log s + s\sqrt{n})
\end{aligned}$$

and taking into account the bound given in Eqn. (8) we obtain

$$\mathsf{E}[\,T\,] \le nm + s\,\min\left\{ m, \frac{\log(2s)}{2} + \pi\sqrt{\frac{n}{6}} \right\}. \tag{13}$$

A closer inspection of the upper bound (13) reveals that the bound could be strengthened with respect to the additive part $\pi\sqrt{n/6}$ which stems from the generality of Theorem 1. If the distribution of the random variables are taken into account then the bound for the maximum will become more accurate. We have made 30 independent experiments for each combination of $(n+1) \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$, $s \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 1000\}$, and $m \in \{1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 100\}$. Here, we only present the 10 worst results with regard to absolute (see Table 1) and relative deviation (see Table 2) between the bound in Eqn. (13) and the observed mean.

Table 1. Results of experiments with the ten worst absolute deviations (abs. Δ) between bound and observed mean.

$n+1$	s	m	MIN	MAX	MEAN	BOUND	ABS. Δ	Δ%
1000	100	50	50,149	50,253	50,196.8	54,118.8	3,922.0	7.81
1000	100	100	100,107	100,236	100,166.6	104,068.8	3,902.2	3.90
1000	100	40	40,149	40,282	40,207.3	43,960.0	3,752.7	9.33
1000	90	50	50,111	50,257	50,178.7	53,699.9	3,521.2	7.02
1000	90	100	100,078	100,243	100,157.2	103,649.9	3,492.7	3.49
1000	90	40	40,111	40,240	40,169.7	43,560.0	3,390.3	8.44
1000	80	50	50,103	50,198	50,140.1	53,281.2	3,141.1	6.26
1000	80	100	100,072	100,178	100,114.4	103,231.2	3,116.8	3.11
1000	80	40	40,095	40,189	40,143.5	43,160.0	3,016.5	7.51
1000	100	30	30,153	30,258	30,202.7	32,970.0	2,767.3	9.16

Table 2. Results of experiments with the ten worst relative deviations (Δ %) between bound and observed mean.

$n+1$	s	m	MIN	MAX	MEAN	BOUND	ABS. Δ	Δ %
10	100	5	170	196	184.5	544.8	360.3	195.29
10	90	5	157	187	171.9	492.8	320.9	186.66
10	80	5	144	172	157.2	441.0	283.8	180.52
10	70	5	132	161	144.6	389.4	244.8	169.33
10	100	4	146	175	164.7	436.0	271.3	164.72
10	60	5	115	145	128.7	338.2	209.5	162.81
10	90	4	141	160	150.7	396.0	245.3	162.77
10	70	4	116	135	124.1	316.0	191.9	154.63
10	80	4	127	153	140.3	356.0	215.7	153.74
20	100	5	215	247	236.8	595.0	358.2	151.27

Finally, we sketch a potential avenue to strengthen the result; its elaboration remains for future work. Recall from the discussion leading to Eqn. (10) that the G_i are geometricly distributed random variables with parameter $p_{i,i+1}$ and that T'_0 is just the sum of the G_i for $i = 1, \ldots, s-1$. Thus, the maximum of $n+1$ samples of T'_0 is the maximum of $n+1$ sums of geometric random variables. Since $\max\{a_1 + b_1, a_2 + b_2\} \leq \max\{a_1, a_2\} + \max\{b_1, b_2\}$ we obtain an upper bound by the sum over the maxima of $s-1$ i.i.d. (!) geometric random variables. Unfortunately, the expectation of the maximum of geometric random variables cannot be determined exactly, in contrast to its minimum. But we can use the asymptotic theory of extreme value distributions [3] for getting some evidence that the maximum increases by order $\log(n)\,\mathsf{D}[\,T'_0\,]$ rather than order $\sqrt{n}\,\mathsf{D}[\,T'_0\,]$. Thus, we *conjecture* that

$$\mathsf{E}[\,T\,] = \mathcal{O}(n\,m + s\,\min\{m, \log s + \log n\})\,.$$

3.2 Bi-directional Ring Topology

The modifications of the results required in case of subpopulations at the vertices of a ring with bi-directional migration paths are straightforward: It takes $(n+1)\,m/2$ generations until an individual from each of the two possible migration paths arrive at the last vertex if n is odd (i.e., if the number of subpopulations is even). Therefore the upper bounds are

$$\mathsf{E}[\,T\,] \leq \frac{(n+1)\,m}{2} + \max\{T'_0, T'_1, \ldots, T'_n\}$$

and

$$\mathsf{E}[\,T\,] \leq \frac{(n+1)\,m}{2} + m\,s\,.$$

In the following we can use the same arguments and bounds as those from the preceding subsection.

3.3 Almost Arbitrary Connected Topology

Let $\mathcal{G} = (V, E)$ denote the directed graph describing the migration paths between subpopulations. Needless to say, we assume that the graph is connected and that each vertex can be reached from any other vertex of the graph. As the preceding two examples have shown, the takeover time can be bounded by the time to reach each vertex in the graph (which is bounded by the *diameter* of the graph) plus the time required for takeover in $n+1$ parallel subpopulations. Consequently, the expected takeover time of (almost) arbitrary graphs can be bounded by the two bounds

$$\mathsf{E}[\,T\,] \le \text{diam}(\mathcal{G})\, m + \max\{T'_0, T'_1, \ldots, T'_n\}$$

and

$$\mathsf{E}[\,T\,] \le \text{diam}(\mathcal{G})\, m + m\, s\,.$$

Of course, these bounds can be improved if more information about a graph is known. For example, if we have a d-regular bi-directional graph then at least one best copy enters the population initially, d best copies will leave at the next migration event, and from now on d copies of the best individual will enter the subpopulation at each migration event.

4. Conclusions

It has been shown that the takeover time in parallel populations with migration is bounded by the diameter of the migration graph plus the time until takeover in parallel population occurs. These takeover times are dependent on the selection operation deployed in each subpopulation. Here, we have developed bounds for a particular non-generational selection method. It is conjectured that the bounds can be improved considerably as soon as a sufficiently tight bound for $\mathsf{E}[\,\max\{T'_0, T'_1, \ldots, T'_n\}\,]$ has been developed. In case of non-generational selection methods an appropriate bound for the maximum of geometrically distributed random variables is required. These tasks and the development of tight lower bounds will be part of future work.

References

[1] E. Alba and A. Luque. Growth curves and takeover time in distributed evolutionary algorithms. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2004)*, pages 864–876, Seattle, WA, USA, 2004.

[2] U. Chakraborty, K. Deb, and M. Chakraborty. Analysis of selection algorithms: A Markov chain approach. *Evol. Comput.*, 4(2):133–167, 1996.

[3] H.A. David. *Order Statistics*. 2nd edition, Wiley, New York, 1981.

[4] M. Giacobini, M. Tomassini, and A. Tettamanzi. Modeling selection intensity for linear cellular evolutionary algorithms. In *Proc. 6th International Conference on Artificial Evolution (EA'03)*, Marseille, France, 2003.

[5] M. Giacobini, E. Alba, M. Tomassini, and A. Tettamanzi. Modeling selection intensity for toroidal cellular evolutionary algorithms. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2004)*, pages 1138–1149, Seattle, WA, USA, 2004.

[6] M. Giacobini, M. Tomassini, and A. Tettamanzi. Takeover time curves in random and small-world structured populations. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2005)*, pages 1333–1340, Washington, DC, USA, 2005.

[7] D.E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, Morgan Kaufmann, San Mateo, 1991, pp. 66–93.

[8] G. Rudolph. On takeover times in spatially structured populations: Array and ring. In *Proc. 2nd Asia-Pacific Conference on Genetic Algorithms and Applications*, pages 144–151, Hong Kong, 2000.

[9] G. Rudolph. Takeover times and probabilities of non-generational selection rules. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2000)*, pages 903–910, Las Vegas, NV, USA, 2000.

[10] G. Rudolph. Takeover times of noisy non-generational selection rules that undo extinction. In *Proc. 5th International Conference on Artificial Neural Nets and Genetic Algorithms (ICANNGA 2001)*, pages 268–271, Prague, Czech Republic, 2001.

[11] J. Smith and F. Vavak. Replacement strategies in steady state genetic algorithms: Static environments. In W. Banzhaf and C. Reeves, editors, *Foundations of Genetic Algorithms 5*, Morgan Kaufmann, San Francisco, CA, USA, 1999, pp. 219–233.

REAL-PARAMETER OPTIMIZATION USING STIGMERGY

Peter Korošec, Jurij Šilc
Computer Systems Department
Jožef Stefan Institute, Ljubljana, Slovenia
{peter.korosec,jurij.silc}@ijs.si

Abstract This paper describes the so-called Differential Ant-Stigmergy Algorithm (DASA), which is an extension of the Ant-Colony Optimization for continuous domain. A performance study of the DASA on a benchmark of real-parameter optimization problems is presented. The DASA is compared with a number of evolutionary optimization algorithms including covariance matrix adaptation evolutionary strategy, differential evolution, real-coded memetic algorithm, and continuous estimation of distribution algorithm. The DASA is also compared to some other ant-based methods for continuous optimization. The result obtained indicate a promising performance of the new approach.

Keywords: Ant stigmergy, Benchmark functions, Real-parameter optimization

1. Introduction

Real-parameter optimization is an important issue in many areas of human activities. The general problem is to find a set of parameter values, $\mathbf{x} = (x_1, x_2, \ldots, x_D)$, that minimizes a function, $f(\mathbf{x})$, of D real variables, i.e.,

$$\text{Find: } \mathbf{x}^* \mid f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathbb{R}^D.$$

In the past two or three decades, different kinds of optimization algorithms have been designed and applied to solve real-parameter function optimization problems. Some of the popular approaches are real-parameter genetic algorithms [17], evolution strategies [3], differential evolution [14], particle swarm optimization [8], classical methods such as quasi-Newton method [12], other non-evolutionary methods such as simulated annealing [9], tabu search [6] and lately ant-colony based algorithms.

Algorithms inspired by model of ant colony behavior are increasingly successful among researches in computer science and operational research. A particular successful metaheuristic—Ant-Colony Optimization (ACO)—as a common framework for the existing applications and algorithmic variants of a variety of ant algorithms has been proposed by Dorigo and colleagues [4]. However, a direct application of the ACO for solving real-parameter optimization problem is difficult. The first algorithm designed for continuous function optimization was continuous ant colony optimization (CACO) [2] which comprises two levels: global and local. CACO uses the ant colony framework to perform local searches, whereas global search is handled by a genetic algorithm. Up to now, there are few other adaptations of ACO algorithm to continuous optimization problems: continuous interacting ant colony (CIAC) [5], ACO for continuous and mixed-variable (eACO) [13], and agregation pheromone system [16].

In this paper we will introduce a new approach to the real-parameter optimization problem using an ACO-based algorithm that uses the pheromonal trail laying—a case of *stigmergy*—as a means of communication between ants.

2. The Differential Ant-Stigmergy Algorithm

2.1 The Fine-Grained Discrete Form of Continuous Domain

In the following, a process of transformation from a continuous domain into a fine-grained discrete form is presented.

Let $x'_i(s)$ be the current value of the i-th parameter. During the searching for the optimal parameter value, the new value, x_i, is assigned to the i-th parameter as follows:

$$x_i = x'_i + \delta_i. \tag{1}$$

Here, δ_i is the so-called *parameter difference* and is chosen from the set

$$\Delta_i = \Delta_i^- \cup \{0\} \cup \Delta_i^+,$$

where

$$\Delta_i^+ = \left\{ \delta_{i,k}^+ \mid \delta_{i,k}^+ = b^{k+L_i-1}, k = 1, 2, \ldots, d_i \right\}$$

and

$$\Delta_i^- = \left\{ \delta_{i,k}^- \mid \delta_{i,k}^- = -b^{k+L_i-1}, k = 1, 2, \ldots, d_i \right\}.$$

Here $d_i = U_i - L_i + 1$. Therefore, for each parameter x_i, the parameter difference, δ_i, has a range from b^{L_i} to b^{U_i}, where b is the so-called *discrete base*, $L_i = \lfloor \log_b(\varepsilon_i) \rfloor$, and $U_i = \lfloor \log_b(\max(x_i) - \min(x_i)) \rfloor$. With the parameter ε_i, the maximum precision of the parameter x_i is set. The precision is limited by the computer's floating-point arithmetics.

Let us consider a simple example for the parameter x_i with $\max(x_i) = 400$, $\min(x_i) = -350$, $b = 10$, and $\varepsilon_i = 10^{-3}$. Then $L_i = -3$, $U_i = 2$, and $d_i = 6$. Finally, Δ_i is constructed as follows:

$$\left\{-10^2, -10^1, -10^0, -10^{-1}, -10^{-2}, -10^{-3}, 0, 10^{-3}, 10^{-2}, 10^{-1}, 10^0, 10^1, 10^2\right\}.$$

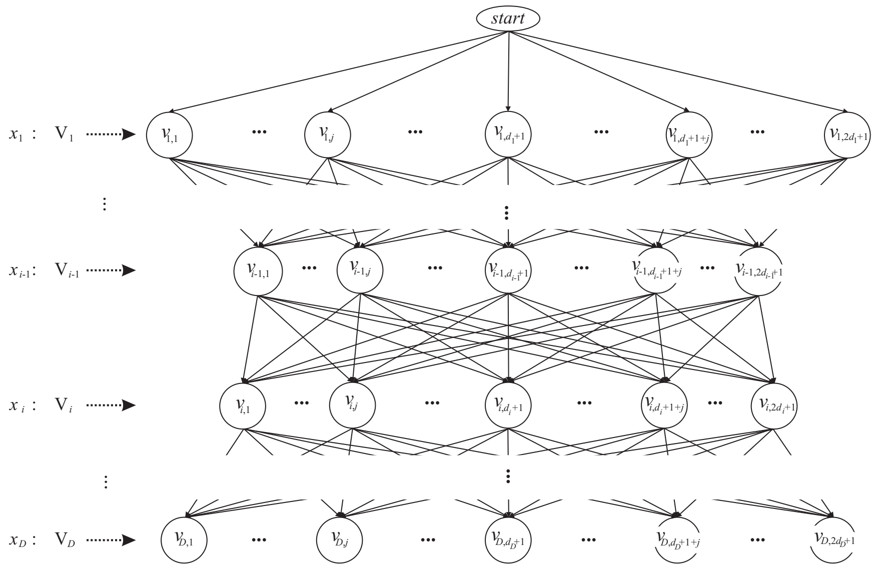


Figure 1. Differential graph.

2.2 Differential Graph

From all the sets Δ_i, $1 \leq i \leq D$, a so-called *differential graph* $\mathcal{G} = (V, E)$ with a set of vertices, V, and a set of edges, E, between the vertices is constructed. Each set Δ_i is represented by the set of vertices, $V_i = \{v_{i,1}, \ldots, v_{i,2d_i+1}\}$, and $V = \bigcup_{i=1}^{D} V_i$. Then we have that

$$\Delta_i = \{\underbrace{\delta^-_{i,d_i}, \ldots, \delta^-_{i,d_i-(j-1)}, \ldots,}_{\Delta_i^-} 0, \underbrace{\ldots, \delta^+_{i,j}, \ldots, \delta^+_{i,d_i}}_{\Delta_i^+}\}$$

is equal to

$$V_i = \{v_{i,1}, \ldots, v_{i,j}, \ldots, \underbrace{v_{i,d_i+1}}_{0}, \ldots, v_{i,d_i+1+j}, \ldots, v_{i,2d_i+1}\},$$

where $1 \leq j \leq d_i$. Each vertex of the set V_i is connected to all the vertices that belong to the set V_{i+1} (see Figure 1). Therefore, this is a directed graph, where each path from the $start$ vertex to any of the ending vertices is of equal length and can be defined with v_i as:

$$v = v_1 v_2 \cdots v_D,$$

where $v_i \in V_i$, $1 \leq i \leq D$.

The optimization consists of finding a path υ, such that $f(\mathbf{x}) < f(\mathbf{x}')$, where $\mathbf{x}'$ is currently the best solution, and $\mathbf{x} = \mathbf{x}' + \Delta(\upsilon)$ (using Eqn. (1)). Additionally, if the cost function $f(\mathbf{x})$ is smaller than the $f(\mathbf{x}')$, then $\mathbf{x}'$ values are replaced with $\mathbf{x}$ values.

To enable a more flexible movement over the search space, the weight ω is added to Eqn. (1):

$$x_i = x_i' + \omega\delta_i, \tag{2}$$

where $\omega = random(0, b)$ and $\omega = 1, 2, \ldots, b-1$.

2.3 Search Algorithm

The optimization consists of an iterative improvement of the currently best solution, $\mathbf{x}'$, by constructing an appropriate path υ, which with the use of Eqn. (2) returns a new best solution, and is done as follows:

1 A solution $\mathbf{x}'$ is randomly chosen.

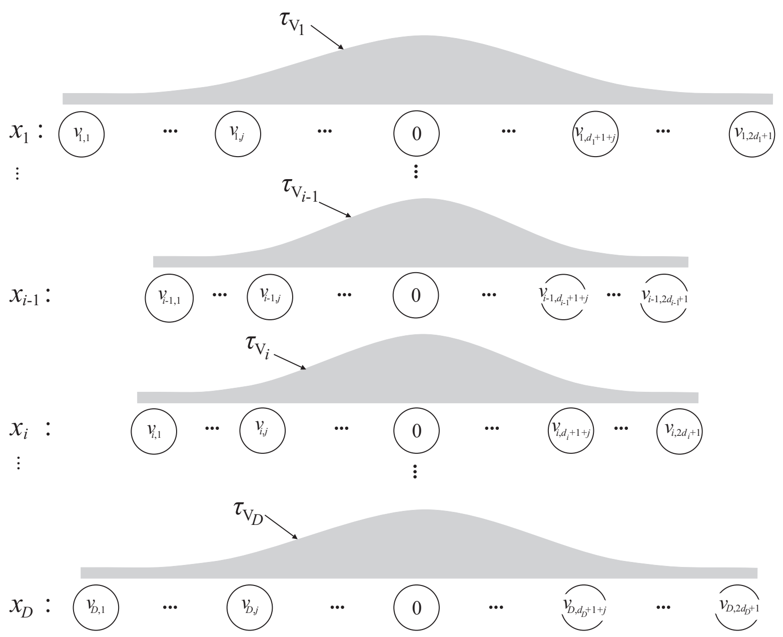


Figure 2. Initial pheromone distribution.

2 An initial amount of pheromone, $\tau_{V_i}^0$, is deposited on all the vertices from the set $V_i \subset V, 1 \leq i \leq D$, according to a Gaussian probability density function

$$g(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where μ is the mean, σ is the standard deviation, and $\mu = 0$, $\sigma = 1$ (see Figure 2).

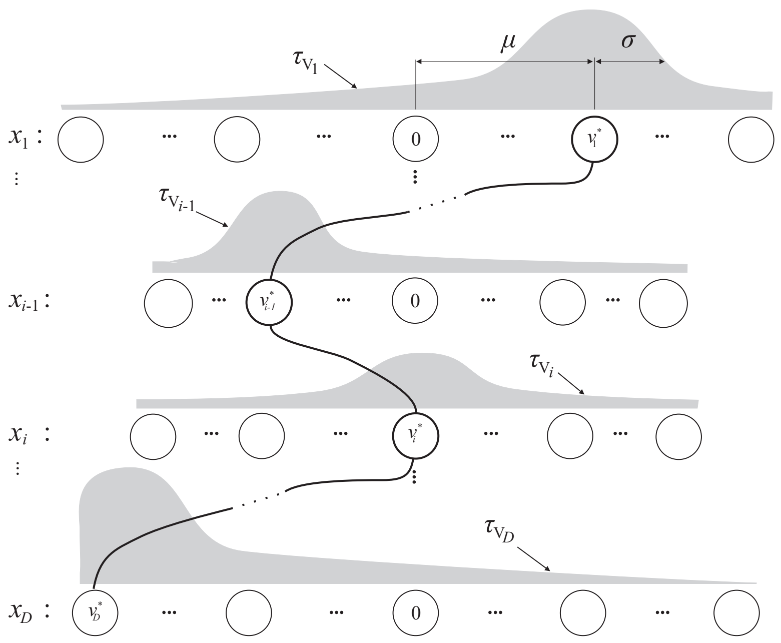


Figure 3. Pheromone distribution after a new best solution is found.

3 There are m ants in a colony, all of which begin simultaneously from the $start$ vertex. The probability with which they choose the next vertex depends on the amount of pheromone on the vertices. Ants use a probability rule to determine which vertex will be chosen next. More specifically, ant α in step i moves from a vertex in set V_{i-1} to vertex $v_{i,j} \in \{v_{i,1}, \ldots, v_{i,2d_i+1}\}$ with a probability given by:

$$p_j(\alpha, i) = \frac{\tau(v_{i,j})}{\sum_{1 \le k \le 2d_i+1} \tau(v_{i,k})}, \tag{3}$$

where $\tau(v_{i,k})$ is the amount of pheromone on vertex $v_{i,k}$. The ants repeat this action until they reach the ending vertex. For each ant, solution $\mathbf{x}$ is constructed (see Eqn. (2)) with a calculation of $f(\mathbf{x})$. The best solution, $\mathbf{x}^*$, out of m solutions is compared to the currently best solution $\mathbf{x}'$. If $f(\mathbf{x}^*)$ is smaller than $f(\mathbf{x}')$, then $\mathbf{x}'$ values are replaced with $\mathbf{x}^*$ values. Furthermore, in this case the pheromone amount is redistributed according to the associated path $v^* = v_1^* \cdots v_{i-1}^* v_i^* \cdots, v_D^*$. New probability density functions have maxima on the vertices v_i^*, and the standard deviations are inversely proportioned to the solution's improvement (see Figure 3).

4 The amount of pheromone is distributed by some predetermined percentage, ρ, on each probability density function as follows:

$$\mu \leftarrow (1-\rho)\mu \quad \text{and} \quad \sigma \leftarrow (1+\rho)\sigma.$$

Pheromone dispersion has a similar effect to pheromone evaporation in the classical ACO algorithm.

5 The whole procedure is then repeated until some ending condition is met.

We named the search algorithm presented in this section as the *Differential Ant-Stigmergy Algorithm* (DASA).

3. Performance Evaluation

3.1 The Experimental Environment

The platform used to perform the experiments was based on AMD Opteron$^{\text{TM}}$2.6-GHz processor, 2 GB of RAM, and the Microsoft®Windows®XP operating system.

The DASA has only three parameters: the number of ants, m, the pheromone disperse factor, ρ, and the maximum parameter precision, ε. Their settings are: $m = 10$, $\rho = 0.1$, and $\varepsilon = 10^{-12}$.

3.2 The Benchmark Suite

The DASA was investigated on the four test functions for dimension 30. The complete definition of the test-suit is available in [15]. Function f_3 (*Shifted Rotated High Conditional Eliptic Function*) is unimodal and function f_9 (*Shifted Rastrigin's Function*) is multi-modal. Functions f_{13} (*Expanded Extended Griewank's plus Rosenbrock's Function*) and f_{15} (*Hybrid Composition Function*) result from the composition of several functions. To prevent exploitation of the symmetry of the search space and of the typical zero value associated with the global optimum, the local optimum is shifted to a value different from zero, and the function the function values of the global optimum are non zero.

3.3 A Comparison of Algorithms

The DASA was compared to four well-known algorithms:

A restart *Covariance Matrix Adaptation Evolution Strategy* with increasing population size (CMA-ES) [1]: The CMA-ES introduced by Hansen and Ostermeier [7] is an evolutionary strategy that adapts the full covariance matrix of a normal search (mutation) distribution. By increasing the population size for each restart—as is suggested in [1]—the search characteristics become more global after each restart.

A *Differential Evolution* (DE) [11]: DE is a stochastic, population-based optimization algorithm. It was introduced by Storn and Price [14] and was developed to optimize the real (float) parameters of a real-valued function. DE resembles the structure of an evolutionary algorithm, but differs from traditional evolutionary algorithms in its generation of new candidate solutions and by its use of a 'greedy' selection scheme.

A real-coded *Memetic Algorithm* (MA) [10]: The MA is a genetic algorithm (GA) that applies a separate local search (LS) process to refine new individuals. The GA applied to make the exploration (i.e., to maintain diversity in the population), the LS applied to improve new solutions (i.e., to exploit the most promising regions of the domain search). In [10] a steady-state GA is used.

A continuous *Estimation of Distribution Algorithm* (EDA) [18]: The EDA is based on probabilistic modeling instead of classical genetic operators such as crossover or mutation. The EDA used in [18] employs a multivariate Gaussian distribution and is therefore able to represent a correlation between variables in the selected individuals via the full covariance matrix of the system.

3.4 The Evaluation

The function error, $f(\mathbf{x}) - f(\mathbf{x}^*)$ with $\mathbf{x}^*$ being the optimum, is recorded at four checkpoints (1,000, 10,000, 100,000, and 300,000 function evaluations). The error data is collected for 25 runs after which the trials are ordered from best to worst. The trial mean and the standard deviation as well as the results of the best, median, and worst trail are presented for each of the four checkpoints. The error values are presented in Tables 1 and 2.

The convergence rates of the DASA on functions f_3, f_9, f_{13}, and f_{15} are plotted in Figure 4. The rates show the median performance of the 25 runs. In the figure, the function error is plotted against the number of evaluations.

3.5 The Complexity of the Algorithm

To estimate the algorithm's complexity we have calculated $\frac{\widehat{T_2}-T_1}{T_0}$, where T_0 is independent of the function dimension and is calculated by running the program:

```
for i = 1 to 1,000,000
    x = (double) 5.55; x = x + x;
    x = x * x; x = sqrt(x);
    x = ln(x); x = exp(x);
    y = x/x
end
```

T_1 is the computing time for 200,000 evaluations only for function f_3, and $\widehat{T_2}$ is the mean of five executions, but now considering the complete computing time of the algorithm for the function f_3. The results are included in Table 3.

Table 1. Error values for the thirty-dimensional f_3 and f_9, measured after 1,000, 10,000, 100,000, and 300,000 function evaluations.

Function evaluations		Algorithm				
		CMA-ES	DE	MA	EDA	DASA
f_3						
1e3	*Best*	3.84 e+08	2.18 e+08	9.63 e+07	8.95 e+08	6.11 e+07
	Median	1.00 e+09	5.66 e+08	2.69 e+08	1.23 e+09	2.80 e+08
	Worst	2.07 e+09	9.53 e+08	5.82 e+08	1.92 e+09	5.57 e+08
	Mean	1.07 e+09	5.53 e+08	2.94 e+08	1.25 e+09	3.10 e+08
	Std	4.43 e+08	1.78 e+08	3.04 e+07	2.67 e+08	1.31 e+08
1e4	*Best*	1.24 e+06	3.58 e+07	1.81 e+07	1.79 e+08	4.55 e+06
	Median	4.90 e+06	6.90 e+07	4.17 e+07	2.71 e+08	1.15 e+07
	Worst	1.42 e+07	1.66 e+08	8.51 e+07	3.84 e+08	1.95 e+07
	Mean	6.11 e+06	8.15 e+07	4.14 e+07	2.76 e+08	1.16 e+07
	Std	3.79 e+06	3.25 e+07	2.95 e+06	5.03 e+07	4.44 e+06
1e5	*Best*	4.07 e−09	3.89 e+05	1.76 e+06	2.41 e+07	5.77 e+05
	Median	5.44 e−09	1.33 e+06	4.91 e+06	3.55 e+07	1.07 e+06
	Worst	8.66 e−09	3.38 e+06	6.80 e+06	4.55 e+07	1.94 e+06
	Mean	5.55 e−09	1.52 e+06	5.51 e+06	3.49 e+07	1.23 e+06
	Std	1.09 e−09	8.92 e−05	6.05 e+05	4.94 e+06	3.97 e+05
3e5	*Best*	4.07 e−09	5.46 e+04	5.55 e+05	2.27 e+06	1.27 e+05
	Median	5.44 e−09	2.43 e+05	7.64 e+05	3.66 e+06	4.32 e+05
	Worst	8.66 e−09	9.00 e+05	1.56 e+06	5.88 e+06	8.15 e+05
	Mean	5.55 e−09	2.89 e+05	8.77 e+05	3.75 e+06	4.59 e+05
	Std	1.09 e−09	1.93 e+05	5.81 e+04	9.09 e+05	2.02 e+05
f_9						
1e3	*Best*	2.19 e+02	2.99 e+02	1.82 e+02	4.07 e+02	4.60 e+01
	Median	2.50 e+02	3.72 e+02	3.00 e+02	4.76 e+02	9.13 e+01
	Worst	2.87 e+02	4.25 e+02	4.00 e+02	5.44 e+02	1.52 e+02
	Mean	2.53 e+02	3.77 e+02	2.99 e+02	4.80 e+02	9.29 e+01
	Std	1.65 e+01	3.00 e+01	1.00 e+01	3.51 e+01	2.75 e+01
1e4	*Best*	2.39 e+01	8.17 e+01	6.28 e+01	3.23 e+02	9.95 e−01
	Median	4.88 e+01	9.74 e+01	1.04 e+02	3.66 e+02	2.99 e+00
	Worst	7.96 e+01	1.13 e+02	1.50 e+02	3.87 e+02	4.98 e+00
	Mean	4.78 e+01	9.85 e+01	1.05 e+02	3.62 e+02	2.95 e+00
	Std	1.15 e+01	8.42 e+00	3.17 e+00	1.62 e+01	1.17 e+00
1e5	*Best*	2.98 e+00	1.90 e−08	3.98 e+00	2.18 e+02	0.00 e+00
	Median	6.96 e+00	5.93 e−08	7.96 e+00	2.50 e+02	0.00 e+00
	Worst	1.19 e+01	1.39 e−07	1.19 e+01	2.78 e+02	0.00 e+00
	Mean	6.89 e+00	6.68 e−08	7.55 e+00	2.50 e+02	0.00 e+00
	Std	2.22 e+00	3.39 e−08	5.36 e−01	1.34 e+01	0.00 e+00
3e5	*Best*	4.35 e−06	0.00 e+00	7.78 e−09	2.10 e+02	0.00 e+00
	Median	9.95 e−01	0.00 e+00	9.95 e−01	2.30 e+02	0.00 e+00
	Worst	4.97 e+00	0.00 e+00	1.99 e+00	2.48 e+02	0.00 e+00
	Mean	9.38 e−01	0.00 e+00	6.81 e−01	2.30 e+02	0.00 e+00
	Std	1.18 e+00	0.00 e+00	1.21 e−01	9.44 e+00	0.00 e+00

Table 2. Error values for the thirty-dimensional f_{13} and f_{15}, measured after 1,000, 10,000, 100,000, and 300,000 function evaluations.

Function evaluations		Algorithm				
		CMA-ES	DE	MA	EDA	DASA
f_{13}						
1e3	*Best*	3.05 e+01	3.12 e+04	4.09 e+02	4.66 e+05	1.33 e+04
	Median	7.36 e+01	1.29 e+05	3.86 e+03	7.39 e+05	1.38 e+05
	Worst	4.98 e+02	4.33 e+05	1.06 e+04	1.13 e+06	6.47 e+05
	Mean	1.14 e+02	1.62 e+05	3.95 e+03	7.50 e+05	2.12 e+05
	Std	1.07 e+02	8.66 e+04	4.62 e+02	1.93 e+05	1.81 e+05
1e4	*Best*	2.46 e+00	3.20 e+01	9.97 e+00	1.35 e+05	2.57 e+00
	Median	3.87 e+00	8.02 e+01	1.49 e+01	2.97 e+05	6.34 e+00
	Worst	5.62 e+00	2.47 e+02	1.96 e+01	5.24 e+05	1.38 e+01
	Mean	3.80 e+00	1.02 e+02	1.51 e+01	3.08 e+05	7.02 e+00
	Std	7.27 e−01	6.33 e+01	4.49 e−01	1.14 e+05	3.33 e+00
1e5	*Best*	2.43 e+00	2.31 e+00	2.76 e+00	1.84 e+03	1.20 e+00
	Median	2.83 e+00	3.90 e+00	9.07 e+00	4.30 e+03	2.02 e+00
	Worst	3.67 e+00	1.39 e+01	1.28 e+01	9.93 e+03	2.73 e+00
	Mean	2.89 e+00	4.55 e+00	8.66 e+00	4.52 e+03	2.04 e+00
	Std	3.59 e−01	2.25 e+00	4.42 e−01	1.91 e+03	4.17 e−01
3e5	*Best*	1.10 e+00	2.31 e+00	1.33 e+00	3.82 e+01	9.62 e−01
	Median	2.61 e+00	3.89 e+00	2.54 e+00	6.86 e+01	1.93 e+00
	Worst	3.20 e+00	1.39 e+01	1.03 e+01	1.29 e+02	2.56 e+00
	Mean	2.49 e+00	4.51 e+00	3.96 e+00	7.36 e+01	1.88 e+00
	Std	5.13 e−01	2.26 e+00	5.38 e−01	2.36 e+01	3.99 e−01
f_{15}						
1e3	*Best*	4.93 e+02	8.82 e+02	5.46 e+02	1.03 e+03	2.32 e+02
	Median	6.93 e+02	1.08 e+03	7.49 e+02	1.14 e+03	6.28 e+02
	Worst	8.51 e+02	1.19 e+03	1.05 e+03	1.21 e+03	7.84 e+02
	Mean	6.69 e+02	1.08 e+03	7.62 e+02	1.13 e+03	5.89 e+02
	Std	1.15 e+02	7.13 e+01	2.64 e+01	4.50 e+01	1.50 e+02
1e4	*Best*	2.08 e+02	6.17 e+02	3.72 e+02	5.90 e+02	4.17 e−04
	Median	4.00 e+02	6.88 e+02	4.30 e+02	6.31 e+02	3.05 e+02
	Worst	5.53 e+02	8.36 e+02	5.42 e+02	8.82 e+02	5.00 e+02
	Mean	3.87 e+02	7.04 e+02	4.41 e+02	6.88 e+02	2.40 e+02
	Std	8.48 e+01	6.30 e+01	7.96 e+00	9.93 e+01	1.59 e+00
1e5	*Best*	2.00 e+02	5.03 e+02	2.00 e+02	4.85 e+02	0.00 e+00
	Median	2.00 e+02	5.18 e+02	3.00 e+02	4.89 e+02	3.00 e+02
	Worst	3.20 e+02	6.33 e+02	5.00 e+02	6.71 e+02	5.00 e+02
	Mean	2.25 e+02	5.20 e+02	3.56 e+02	5.38 e+02	2.33 e+02
	Std	4.10 e+01	2.39 e+01	1.51 e+01	7.66 e+01	1.58 e+02
3e5	*Best*	2.00 e+02	4.75 e+02	2.00 e+02	4.35 e+02	0.00 e+00
	Median	2.00 e+02	4.81 e+02	3.00 e+02	4.59 e+02	3.00 e+02
	Worst	3.00 e+02	5.86 e+02	5.00 e+02	5.63 e+02	5.00 e+02
	Mean	2.08 e+02	4.84 e+02	3.56 e+02	4.81 e+02	2.33 e+02
	Std	2.75 e+01	2.14 e+01	1.51 e+01	4.67 e+01	1.58 e+02

3.6 Comparison to Other Ant Methods

As we mentioned in the introduction, there are few other adaptations of the ACO algorithm to real-parameter optimization. Here, the DASA is compared to results presented by Socha in [13]. In order to have comparable results, the same accuracy level was chosen.

The results presented in Table 4 are based on 25 independent runs of the the DASA and show number of function evaluations to achieve the fixed accuracy level. The experimental results show that the DASA has much higher convergence speed than that of the CACO and comparable with the eACO.

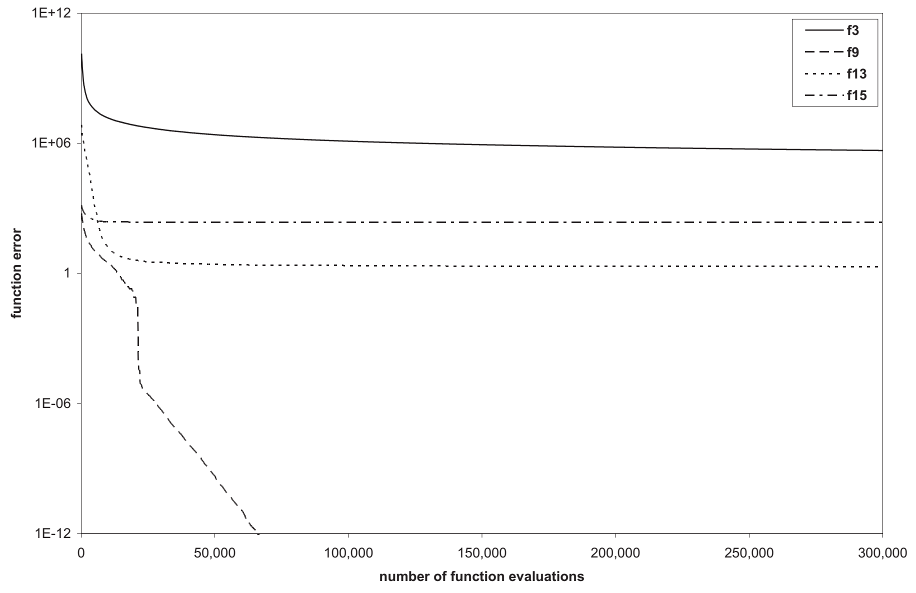


Figure 4. Convergence graph.

4. Discussion and Conclusion

We proposed an extension of the ant-colony optimization metaphor for continuous domain. This new approach was named Differential Ant-Stigmergy Algorithm and was studied on a set of benchmark functions of real-parameter optimization problems.

The algorithm was compared with a number of evolutionary optimization algorithms including covariance matrix adaptation evolutionary strategy, differential evolution, real-coded memetic algorithm, and continuous estimation of distribution algorithm.

Table 3. Algorithm complexity (function f_3, $D = 30$).

ALGORITHM	THE SYSTEM	T_0	T_1	$\widehat{T}_2$	$\frac{\widehat{T}_2 - T_1}{T_0}$
CMA-ES	Pentium 4 3GHz / 1GB Red Hat Linux 2.4 MATLAB 7.0.1	0.40	41.00	*24.00	—
DE	AMD Sempron 2800+ / 1GB Mandrake Linux 10.1 C	0.29	7.64	8.49	2.94
MA	Pentium 4 2.8GHz / 512MB Linux kernel v. 2.6 C++ with GCC 3.3.2	0.42	8.63	13.45	11.48
EDA	Xeon 2.4GHz / 1GB Windows XP (SP2) MATLAB 6	**6.93	1.45	5.22	0.54
DASA	AMD Opteron 2.6GHz / 2GB Windows XP (SP 2) Delphi 2006	0.19	58.94	59.20	1.37

* The large number of T_1 reflect the large number of objective function calls, while for T_2 a complete, eventually large, population is evaluated (serially) within a single function call.
** Due to poor loop implementation in MATLAB 6.

Table 4. Comparison of average number of function evaluations until the accuracy is reached.

TEST FUNCTION*	D	ACCURACY	CACO [2]	CIAC [5]	eACO [13]	DASA
Sphere	6	10^{-4}	22,050	50,000	695	832
Goldstein & Price	2	10^{-4}	5,320	23,391	364	991
Rosenbrock	2	10^{-3}	6,842	11,797	2,905	137
Zakharov	2	10^{-4}	—	—	401	182

* http://iridia.ulb.ac.be/∼ksocha/extaco04.html

The result obtained indicate a promising performance of the new approach. One can notice that our approach performs better then the rest of the approaches on three out of four test functions. Since selected test functions reflected different kinds of pseudo-real optimization problems, one can conclude that the DASA is applicable to many real-parameter optimization problems.

Regarding the future, one important issue consists of pure continuous ant-stigmergy algorithm. Here, so-called parameter differences will be in continuous form instead of fine-grained discrete form.

References

[1] A. Auger and N. Hansen. A Restart CMA Evolution Strategy with Increasing Population Size. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2005)*, Edinburgh, UK, 2005.

[2] G. Bilchev and I.C. Parmee. The Ant Colony Metaphor for Searching Continuous Design Spaces. *Lect. Notes Comp. Sc.*, 993:25–39, 1995.

[3] K. Deb, A. Anand, and D. Joshi. A Computationally Efficient Evolutionary Algorithm for Real-Parameter Optimization. *Evol. Comput.*, 10(4):371–395, 2002.

[4] M. Dorigo and T. Stützle. *Ant Colony Optimization.* The MIT Press, Cambridge, Massachusetts, 2004.

[5] J. Dréo and P. Siarry. A New Ant Colony Algorithm Using the Heterarchical Concept Aimed at Optimization of Multiminima Continuous Functions. *Lect. Notes Coput. Sc.*, 2463:216–227, 2002.

[6] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, Boston, 1997.

[7] N. Hansen and A. Ostermeier. Adapting Arbitrary Normal Mutation Distribution in Evolutionary Strategies: The Covariance Matrix Adaptation. In *Proc. IEEE International Conference on Evolutionary Computation (ICEC 1996)*, pages 312–317, Nagoya, Japan, 1996.

[8] J. Kennedy and R.C. Eberhart. Particle Swarm Optimization. In *Proc. IEEE International Conference on Neural Networks*, pages 1942-–1948, Perth, Australia, 1995.

[9] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 22671–680, 1983.

[10] D. Molina, F. Herrera, and M. Lozano. Adaptive Local Search Parameters for Real-Coded Memetic Algorithms. In *Proc. EEE Congress on Evolutionary Computation (CEC 2005)*, Edinburgh, UK, 2005.

[11] J. Rönkkönen, S. Kukkonen, and K.V. Price. Real-Parameter Optimization with Differential Evolution. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2005)*, Edinburgh, UK, 2005.

[12] G.V. Reklaitis, A. Ravindran, and K.M. Ragsdell. *Engineering Optimization Methods.* Wiley, New York, 1983.

[13] K. Socha. ACO for Continuous and Mixed-Variable Optimization. *Lect. Notes Coput. Sc.*, 3172:25–36, 2004.

[14] R. Storn and K.V. Price. Differential Evolution – A Fast and Efficient Heuristic for Global Optimization over Continuous Space. *J. Global Opt.*, 11(4):341–359, 1997.

[15] P.N. Sunganthan, N. Hansen, J.J. Liang, Y.-P. Chen, A. Auger, and S. Tiwari. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Technical Report, Nanyang Technological University, Singapore, May 2005.

[16] S. Tsutsui. Ant Colony Optimization for Continuous Domain with Aggregation Pheromones Metaphor. In *Proc. 5th International Conference on Recent Advances in Soft Computing*, 2004.

[17] A.H. Wright. Genetic Algorithms for Real Parameter Optimization. In *Proc. 1st Workshop on Foundations of Genetic Algorithms*, pages 205–218, Bloomington, Indiana, USA 1990.

[18] B. Yuan and M. Gallagher. Experimental Results for the Special Session on Real-Parameter Optimization at CEC 2005: A Simple, Continuous EDA. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2005)*, Edinburgh, UK, 2005.

DIETARY MENU PLANNING BY EVOLUTIONARY COMPUTATION

Barbara Koroušić Seljak
Computer Systems Department
Jožef Stefan Institute, Ljubljana, Slovenia
barbara.korousic@ijs.si

Abstract We present an evolutionary computation method for optimal planning of dietary menus, considering nutrient and non-nutrient requirements, and aesthetic standards. The method is based on the Elitist Non-Dominated Sorting Genetic Algorithm and implemented in a multi-level way. The main idea behind the method is to optimize meals and daily menus independently guiding the optimization to the overall Pareto optimal weekly menus. As a demonstration, we applied the method to a weekly-menu planning example: optimization of an existing weekly menu for people without specific dietary requirements in a local hospital.

Keywords: Dietary computer-based menu planning, Multi-level optimization, Multi-objective and multi-constrained evolutionary optimization, Repair methods

1. Introduction

In 2001, a 30-year-old Slovene man of height 177 cm weighed on average 84 kg and a 30-year-old Slovene woman of height 165 cm weighed on average 68.4 kg [3]. These data show that 30-year-old Slovene has on average body mass index (BMI, kg/m^2) greater than 25, which means an elevated risk of developing chronic diseases, such as cardiovascular diseases, diabetes, cancer, osteoporosis, etc.

There are several reasons for overweightness (BMI greater than 25) and obesity (BMI greater than 30), and they have to be considered from different viewpoints. Using a computer program to handle numerous nutrient information and plan menus in a personalized way is one of them.

In this paper, we present an evolutionary computation approach to dietary menu planning that has been applied within a nutrition software [9]. In Section 2, we describe the problem of menu planning; in Section 3, we introduce the evolutionary approach; and finally, in Section 4, we give an evaluation of the approach. We conclude the paper in Section 5, where we list our conclusions and suggest possible future work.

2. Dietary Menu Planning

The problem of dietary menu planning is an intractable optimization problem, because of many constraints and objectives dictated by nutrient and non-nutrient requirements, and aesthetic standards.

It can be formulated as a linear-programming problem because the objectives are specified as linear functions and the constraints are specified as linear equalities or inequalities. A simplified version of the problem, considering basic nutrient requirements and one objective of cost, was firstly solved using a calculator in 1941 [6]. Since then the linear programming methods have improved significantly, producing cost-optimized menus. However, difficulties have been encountered in using numerical representations for qualitative factors, such as taste, consistency, color, temperature, shape, and method of preparation.

We applied the *Elitist Non-Dominated Sorting Genetic Algorithm* (NSGA-II) [4]] in a multi-level way [7] to generate dietary menus, considering *constraints* on nutrient and non-nutrient requirements and *objectives* of low cost, high seasonal quality and functionality, and low deviations from uniformly distributed aesthetic standards for taste, consistency, color, temperature, shape, and method of preparation.

2.1 Mathematical Formulation of the Problem

Mathematically, dietary menu planning reduces to a multi-objective and multi-constrained (multi-dimensional) knapsack problem (MDKP) that is easy to formulate, yet its decision problem is NP-complete. It means that only by using a heuristic optimization method a solution can be found quickly (in a polynomial time).

We define the problem as follows: *Given food items of different values and volumes, find the most valuable composition that fits in a knapsack of fixed volumes. Values are defined subjectively with respect to food functionality, seasonal availability, cost, taste, consistency, color, temperature, shape and method of preparation. Knapsack volumes are defined by the weakly correlated diet-planning principles.*

Food items are selected from a database that integrates nutritional data of more than 7,000 (national and world-wide) foods. We consider the D-A-CH diet-planning principles established by the European nutrition societies [5]. Many other real-world problems can be formulated as a MDKP, for example, the capital budgeting problem, allocating processors in a distributed computer system, project selection, cutting stock problem, etc.

2.2 Multi-Dimensional Knapsack Problem

We are given a knapsack of m volumes $C_k, k = 1, 2, \ldots, m$, and n food items. Each item i has nine values $v_{ik} \in \mathbb{N}^+, v_{ik} > 0, k = 1, 2, \ldots, 9$, and m volumes $\omega_{ik} \in \mathbb{R}^+, \omega_{ik} > 0, k = 1, 2, \ldots, m$, one for each capacity. We are looking for a composition of t items, $t < n$, such that $\sum_{i=1}^{t} \omega_{ik} x_i \Phi C_k$ (Φ can be $\leq$ or $\geq$, $k = 1, 2, \ldots, m$, $t \leq n$), and for which the total values

$$\sum_{i=1}^{t} v_{ik} x_i, k = 1, 2$$

are maximized, while

$$\sum_{i=1}^{t} v_{ik} x_i, k = 3$$

and

$$(\sum_{j=1}^{n_{al}} | \sum_{i=1}^{n} h_{lj}(x_i) - \frac{\sum_{i=1}^{n} h(x_i)}{n_{al}} |) - \sum_{i=1}^{n} h(x_i), l = 4, 5, \ldots, 9$$

are minimized, where n_{al} is the number of possible states of an aesthetic standard l. The functions used in the above objective function are defined as follows:

$$h_{lj}(x_i) = \begin{cases} 0 & x_i = 0 \\ 1 & x_i > 0 \wedge v_{il} = j \end{cases}, \ i = 1, 2, \ldots, n, \ l = 4, 5, \ldots, 9,$$

and

$$h(x_i) = \begin{cases} 0 & x_i = 0 \\ 1 & otherwise \end{cases}, \ i = 1, 2, \ldots, n.$$

The parameter $x_i \in [0.25P_i, 2P_i]$ denotes the quantity of the selected item i expressed in a unit (gram, milligram, microgram, milliliter, etc.). Its value is limited by the fractions of the item's portion size P_i.

2.3 Methods for Solving MDKPs

Exact algorithms that deliver optimum solutions to multi-dimensional knapsack problems in pseudo-polynomial time are based on the branch-and-bound and the dynamic programming approaches. On the other hand, heuristic methods with time complexity bounded by a polynomial in the size parameters of the problem have been known for many decades. A comprehensive review of the multi-constrained 0-1 knapsack problem and the associated heuristic algorithms is given by Chu and Beasley [2]. Some of the ideas are also applicable to non-0-1 MDKPs.

3. Evolutionary Approach to Dietary Menu planning

In our case, a knapsack denotes a weekly menu that is composed of seven consecutive daily menus. By default, each daily menu includes five different meals, i.e., a breakfast, a morning snack, a lunch, an afternoon snack, and a dinner. However, this composition does not bias the method and can be modified to suit the specific menu-planning problem.

We have applied an evolutionary algorithm NSGA-II in a multi-level way. Namely, the problem of weekly-menu planning is logically composed of several smaller sub-problems, one for each daily menu, which have different constraints than the weekly menu. Then, optimization of daily menus is coordinated in order to obtain the overall weekly menu. Further, each daily-menu planning sub-problem is decomposed into several sub-problems of composing courses into meals.

The main idea behind the multi-level method is to optimize each sub-problem independently using a 'local' NSGA-II with the aim to find the overall Pareto-optimal solutions of the problem (i.e., solutions that cannot be improved upon without hurting at least one of the objectives) using the 'global' NSGA-II.

3.1 Encoding

We encode candidate solutions of the weekly menu-planning problem and its sub-problems by integer-valued coding. In our representation, a chromosome at the highest level contains seven data, carrying the information about the daily menus. At the next level, a chromosome contains five data carrying the information about the meals. At the deepest level, a chromosome is formed of a number of pairs $(code_i, x_i)$, where $code_i$ denotes the database code of a food item i and x_i its quantity expressed in grams. By default, the number of pairs varies between 1 and 10, depending on the number of courses (dishes) of the meal.

3.2 Populations

In our implementation, the 'global' NSGA-II starts an evolution from a global population of either random candidate solutions or solutions known from experience. The global population's size is N and remains constant over all generations. Each sub-problem at the next two levels is solved by a 'local' NSGA-II, and operates on its own population of the same size N. Initially, the daily-menu-level and the meal-level local populations are filled with the candidate solutions from the global population and the second-level local populations, respectively.

Beside the global population, we use an additional global pool of candidate solutions that has a function of an archive of the union of solutions generated by the sub-problems. At the daily-menu and the meal level, we use seven and five local pools, respectively, whose function is equal to the global pool's function. Initially, the global and the local pools are empty.

3.3 Fitness Evaluation

In each generation, the fitness of the (global or local) population is evaluated using the following objective functions:

$$f_k(\vec{x}) = \frac{1}{\sum_{i=1}^{n} v_{ik} x_i}, k = 1, 2,$$

$$f_3(\vec{x}) = \sum_{i=1}^{n} v_{i3} x_i,$$

$$f_l(\vec{x}) = (\sum_{j=1}^{n_{al}} |\sum_{i=1}^{n} h_{lj}(x_i) - \frac{\sum_{i=1}^{n} h(x_i)}{n_{al}}|) - \sum_{i=1}^{n} h(x_i), 1 \leq i \leq n,\ 4 \leq l \leq 9, \tag{1}$$

$$h_{lj}(x_i) = \begin{cases} 0 & x_i = 0 \\ 1 & x_i > 0 \wedge v_{il} = j \end{cases},\ 1 \leq i \leq n,\ 4 \leq l \leq 9,$$

$$h(x_i) = \begin{cases} 0 & x_i = 0 \\ 1 & otherwise \end{cases},\ 1 \leq i \leq n.$$

where v_{i1} denotes the functionality of the food item i, v_{i2} its quality in the season, v_{i3} the cost, v_{i4} the taste, v_{i5} the consistency, v_{i6} the color, v_{i7} the temperature, v_{i8} the shape, v_{i9} the method of preparation, and $n_{al}$the number of possibilities for the l-th aestetic standard. The aim of the 'global' and the 'local' evolutionary algorithms is to *minimize* the objective functions of (1).

3.4 Infeasible Solutions

A candidate solution may be highly fit but infeasible if it violates at least one problem constraint. At the deepest level, the constraints for meals are least restrictive:

- Each food item can be selected in a quantity that is limited by its original portion size:

$$g_1(\vec{x}) = x_i \geq 0.25 P_i, \quad g_2(\vec{x}) = x_i \leq 2 P_i. \tag{2}$$

- The energy provided by the meal has to be within the lower limit and the upper limit:

$$g_3(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iE} x_i \geq 0.9E, \quad g_4(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iE} x_i \leq 1.1E, \tag{3}$$

  where ω_{iE} denotes the energy of 100 grams of the food item i, x_i the quantity of the item i expressed in grams, and E the meal requirement for energy.

- The macronutrients (i.e., proteins, lipids and carbohydrates) need to be balanced:

$$\begin{gathered}
g_5(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iP} 4x_i \geq 0.1E, \quad g_6(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iP} 4x_i \leq 0.15E, \\
g_7(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iL} 9x_i \geq 0.15E, \quad g_8(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iL} 9x_i \leq 0.3E, \\
g_9(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iC} 4x_i \geq 0.55E, \quad g_{10}(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iC} 4x_i \leq 0.75E,
\end{gathered} \tag{4}$$

  where ω_{iP}, ω_{iL}, ω_{iC} denote the quantity of proteins, lipids and carbohydrates, respectively, in 100 grams of the food item i, and $N_C$is the number of courses in the meal. Because the quantities are expressed in grams, conversion factors (4 for proteins and carbohydrates, and 9 for lipids) are required to attain to calories. We applied usual balancing factors for adults (0.1 and 0.15 for proteins, 0.15 and 0.3 for lipids, and 0.55 and 0.75 for carbohydrates) but may be changed.

At the upper level, there are additional constraints that need to be satisfied by a feasible chromosome presenting a daily menu:

- Simple sugars should account for only 10 percent or less of the day's total energy intake:

$$g_{11}(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iS} 4x_i \leq 0.1E_d, \tag{5}$$

  where E_d denotes the daily requirement of energy, and ω_{iS} the quantity of simple sugars in 100 g of the food item i.

- The daily intake of saturated fatty acids should be limited to 10 percent of the day's total energy intake:

$$g_{12}(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iF} 9x_i \leq 0.1E_d, \tag{6}$$

where ω_{iF} denotes the quantity of saturated fatty acids in 100 grams of the food item i.

- The recommended daily intake of the dietary fiber is 10 grams per 1000-calorie energy intake and should not exceed 40 grams:

$$g_{13}(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iV} x_i \geq 0.01 E_d, \quad g_{14}(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iV} x_i \leq 40, \qquad (7)$$

where ω_{iV} denotes the quantity of dietary fiber in 100 grams of the food item i.

- The minimum and the maximum sodium requirements for adults in Slovenia are set at 550 and 2400 milligrams per day, respectively [5]:

$$g_{15}(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iN_a} x_i \geq 500, \quad g_{16}(\vec{x}) = \sum_{i=1}^{N_c} \omega_{iN_a} x_i \leq 2400, \qquad (8)$$

where ω_{iN_a} denotes the quantity of sodium in 100 grams of the food item i.

At the highest level, beside the meal and the daily-menu constraints, a chromosome presenting a *weekly menu* has to satisfy all the remaining constraints for nutrients, such as cholesterol, monounsaturated fatty acids, omega-3 and omega-6 polyunsaturated fatty acids, trans-fatty acids, water-soluble and fat-soluble vitamins, water, major minerals, and trace minerals, to become a feasible solution. Formal definitions of these constraints are similar to that of Eqn. (3) or Eqn. (8), but are beyond the scope of this paper.

Repair Method

We decided to repair a certain part of infeasible solutions in each generation to speed up the procedure of finding an optimal solution:

- At the deepest level, we apply a local optimization procedure of *linear programming* trying to convert infeasible solutions into feasible ones. The procedure, based on the simplex method [1], modifies the quantities of randomly selected infeasible chromosome's food items to satisfy the problem constraints.

- At the upper levels, we try to repair infeasible solutions by 'replacing' certain critical meals with more appropriate ones. We apply the *Baldwinian repair*, where replacement is used only to evaluate the fitness values of each solution [8]. Critical meals are those that do not satisfy the constraints on major food groups (i.e., breads, cereal, rice, and pasta / vegetables / fruits / milk, yogurt, and cheese / meat, poultry, fish, beans, eggs, and nuts / fats, oils, and sweets). Namely, a daily menu has to be composed of a certain *number of foods* from each major food group, while a weekly menu has to include a *diverse set of foods* from the major food groups. There may be limitations on frequency of red meat, fish, potato etc.

3.5 Selection

In order to form a new population, a binary tournament approach is applied. Solutions from both - the parent and the previous offspring - populations can take part in the tournament if they are sorted by two attributes, i.e., a *non-domination rank* and a *crowding distance* [4]. Initially, the offspring population is an empty set.

First, solutions are sorted by the fast non-dominated sorting approach of the NSGA-II [4]. In this approach, best non-dominated solutions become elites of identical importance, forming Pareto-optimal fronts. Solutions are non-dominated if none solution is better than the others with respect to all equally important objectives.

Then, solutions are sorted according to their crowding distances. A crowding distance is a measure of the search space around a chosen solution, which is not occupied by any other solution in the population. Its computation requires sorting of the populations according to each objective function value in their ascending order of magnitude. Thereafter, for each objective function, the boundary solutions (solutions with the smallest and the largest function values) are assigned an infinite distance value. All other solutions are assigned a distance value equal to the absolute difference in the function values of two adjacent solutions. This calculation is continued with other objective functions. The overall crowding distance value is calculated as the sum of individual distance values corresponding to each objective.

A solution i wins a tournament with another solution j if both solutions are feasible or infeasible and any of the following conditions are true:

- It has a better non-domination rank than solution j.
- Having the same non-domination rank, it has better crowding distance than solution j.

The first condition makes sure that solution i lies on a better Pareto front than solution j. The second condition resolves the tie of both solutions being on the same non-dominated front by deciding on their crowded distance. The one residing in less crowded area wins. If one solution is feasible and the other is not, the feasible one wins the tournament.

Performing N tournaments, we obtain a new parent population of size N. Other N solutions from the least important Pareto fronts having a smaller crowding distance are discarded.

3.6 Crossover and Mutation

Solutions from the new parent population are mated pair-wise (using a two-point crossover operator) and mutated to create a new offspring population of size N. This completes one NSGA-II iteration.

Mutation is performed on randomly selected elements of the chromosome. The mutation rate is set to be a small value that linearly decreases with iterations. The selected elements are mutated in one of the following ways chosen with respect to the type of the chromosome:

- by replacing a food item or a dish with a food from the same major food group or a dish from the same course group, respectively, or
- by replacing a selected meal with a meal of the same type,
- by replacing a selected daily menu with a daily menu of the same type.

3.7 Termination Criteria

Once a sub-problem (meal planning or daily-menu planning) is solved by a 'local' NSGA-II (using a wanted-solution approach or a time-out approach), its local population is unified with the local populations of the other sub-problems at the same level and saved in their local pool.

To obtain chromosomes at the daily-menu level, meals from a local pool are completed using the rest of the chromosome sequence from the population at this level. The completed solutions (daily menus) are sorted by the non-dominated and the crowding-distance sorting methods to obtain locally optimal solutions, forming a local population of daily menus.

At the weekly-menu level, completed solutions from the local populations of daily menus are unified and saved into a global pool of weekly menus. A selection of optimal solutions (non-dominated solutions with a large crowding distance) from the global pool is transferred to the global population terminating an iteration of the 'global' NSGA-II.

4. Evaluation of the Method

As a demonstration, we applied the multi-level NSGA-II to a problem of planning optimal weekly menus for people without specific dietary requirements in a local hospital. We started the 'global' NSGA-II from an existing non-optimal weekly menu.

In Table 1, we list the parameters used to generate meals, daily menus and weekly menus by the multi-level NSGA-II. We ran the algorithm for 25 times to obtain the experimental results presented in Table 2. In Figure 1, a part of the feasible search space, whose shape is depicted for three objectives, but actually modified by nine objectives, is presented. A subset of the analysis results for a weekly menu generated by the multi-level NSGA-II is presented in Table 3. This weekly menu was generated with respect to the following requirements for the major food group of meat and its substitutes: white meat, legumes, fish and eggs once per week, and red meat three times per week.

Table 1. Parameters.

Parameter	The weekly-menu level	The daily-menu level	The meal level
Chromosomes length	7	5	10
Population size	100	100	100
Pool size	700	500	–
Crossover probability	0.7	0.7	0.7
Mutation probability	0.14–0.01	0.2–0.01	0.1–0.017
Selection type		Two-point crossover	
Crossover type		Linear descending mutation	
Number of iterations	24	18	35

Table 2. Experimental results.

Percentage of infeasible solutions in each new generation	89
Percentage of successfully repaired infeasible solutions	65

	Cost (EUR)	Quality in season	Functionality
Best result	3.08	48	12
Median	9.7	28	6
Worst result	22.8	18	0
Mean value	9.7	28.3	5.8
Standard deviation	3.1	4.7	3.4

5. Conclusions

In this paper, we have presented the NSGA-II in a multi-level way to solve the weekly-menu problem, which is logically decomposed of several sub-problems, namely, daily-menu planning and meal planning. The algorithm finds the Pareto-optimal set of diverse optimal solutions that are trade-offs between high seasonal quality and functionality, and low cost and deviations from the aesthetic standards in a reasonable amount of time. We maintain the feasibility

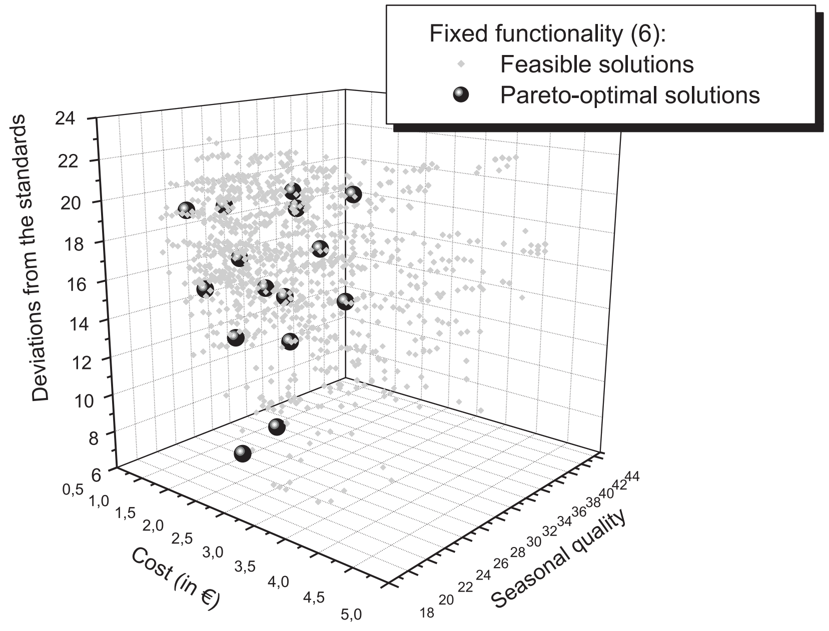


Figure 1. Part of the problem's search space.

Table 3. Analysis results of a computer-generated weekly menu.

	Mean daily values	DACH Recommended Dietary Allowances	Goal achieved (%)
Energy (kcal)	2036	2000	102
Proteins (% of energy)	16	10–15	✓
Lipids (% of energy)	28	15–30	✓
Carbohydrates (% of energy)	56	55–75	✓
Simple sugars (% of energy)	4.5	< 10	✓
Saturated fats (% of energy)	6.6	< 10	✓
Ratio of omega-6 to omega-3 fatty acids	3.9	5	✓
Dietary fibre (g)	33.6	30–40	✓
Cholesterol (mg)	160	300	✓
Sodium (mg)	2,500	550–2,400	104
Breads, cereal, rice, and pasta (no. of units)	11.2	11	102
Vegetables (no. of units)	4.7	5	94
Fruits (no. of units)	3	3	100
Milk, yogurt, and cheese (no. of units)	2	2	100

of solutions by repairing infeasible solutions in two ways, namely, by the LP simplex method (for meals) and the Baldwinian greedy repair method (for daily menus and weekly menus). The experimental results showed that the approach distinguishes with efficiency and effectiveness.

As the problem of dietary menu-planning belongs to the multi-dimensional knapsack problems, the method could be useful for other intractable problems from this group.

Parallel implementation of the multi-level NSGA-II for dietary menu planning deserves future attention.

References

[1] M.A. Bhatti. *Practical Optimization Methods with Mathematica Applications*. Springer-Verlag, New York, 2000.

[2] P.C. Chu and J.E. Beasley. A Genetic Algorithm for the Multidimensional Knapsack Problem. *J. Heuristics*, 4:63–86, 1988.

[3] CINDI Health Monitor, 2001 (in Slovene).

[4] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Ltd., 2001.

[5] *Die Referenzwerte für die Nährstoffzufuhr, D-A-CH Referenzwerte der DGE*. ÖGE, SGE/SVE. 1. Auflage. Umschau Braus Gmbh, Verlagsgesellschaft, Frankfurt/Main, 2002.

[6] E.F. Eckstein. *Menu Planning*. Third Edition. AVI Publishing Company, Westport, Connecticut, 1983.

[7] S. Gunawan, A. Farhang-Mehr, and S. Azarm. Multi-level Multi-objective Genetic Algorithm Using Entropy to Preserve Diversity. *Lect. notes Comput. Sc.*, 2632:148–161, 2003.

[8] H. Ishibuchi, S. Kaige, and K. Narukawa. Comparison between Lamarckian and Baldwinian Repair on Multiobjective 0/1 Knapsack Problems. *Lect. Notes Comput. Sc.*, 3410:370–385, 2005.

[9] Web application for dietary menu planning. `http://optijed.ijs.si`.

OPTIMAL MISSION PLANNING FOR AN AUTONOMOUS UNMANNED AERIAL VEHICLE

Gianpiero Gallo, Giorgio Guglieri, Fulvia B. Quagliotti, Gianluca Speciale
Department of Aeronautical and Space Engineering
Politecnico di Torino, Turin, Italy
gallogianpiero@yahoo.it, {giorgio.guglieri,fulvia.quagliotti,gianluca.speciale}@polito.it

Abstract In this project, we are interested in using computational methods in order to solve the control problem of an unmanned autonomous aerial vehicle. The objective is to have the vehicle navigating in the environment able to reach the desired location through some planned waypoints; this is to be done with the vehicle's best effort, that is with the lowest *cost*. As cost we shall consider miss distance from target waypoint, i.e., a function of the state variables of the mathematical model which describes the dynamics of the vehicle. All this will be done by calculating the optimal trajectory which satisfies all the constraints and contains all the planned waypoints. The optimization part will be done by modifying a micro-genetic algorithm software which was initially developed by David L. Carroll from University of Illinois [3].

Keywords: Aircraft dynamics, Genetic algorithm, Optimal design, Unmanned aerial vehicle

1. Introduction

Computationally efficient trajectory optimization is an enabling technology for many new facets of engineering. Formation flying of satellites, [12], and trajectory generation of unmanned aerial vehicles [11], are two examples where the tools of real-time trajectory optimization would be extremely useful. The capability and roles of Unmanned Aerial Vehicles (UAVs) are evolving, and require new concepts for their control. A significant aspect of this control problem is optimizing the trajectory from the UAV's starting point to its goal. Online trajectory generation for flight control application is important in unmanned aerial vehicles to provide feasible guidance commands in highly aggressive flight situations. In general, the solution of the optimal control problem with high dimensional space is hard to compute. This problem is complicated by the fact that the space of possible control action is extremely large. Two well-known methods that have been applied to this problem are Probabilistic Road Maps [7] (PRMs) and Rapidly-exploring Random Trees [8] (RRTs). These methods reduce the dimensionality of the problem by sampling the possible actions, but the resulting trajectories are generally not optimal. Another different approach to the optimal trajectory problem consist on applying the Model Predictive Control (MPC). MPC refers to a class of algorithms that compute a sequence of manipulated variable adjustments in order to optimize the future behaviour of a system [10]. The main idea of MPC is to choose the control action by repeatedly solving, on-line, an optimal control problem. This aims at minimizing a performance criterion over a future horizon, possibly subject to constraints on the manipulated inputs and outputs, where the future behavior is computed according to a model of the system. An important advantage of MPC is its ability to handle input and state constraints for large scale multivariable plants [1, 2]. Murray [9] has been investigating techniques for generating state and input trajectories which satisfy the equations of motion and trade off tracking performance for inertial stability, using differential flatness.

Stochastic search is an alternative strategy that can bypass some limitations of the previous methods. The genetic algorithms belong to this last family of solvers, as the random choice of the possible solution is combined with criteria for the direction of search which derive from natural evolution of species. This technique is considered global and robust in terms of search over the space of solutions. The genetic algorithm [5] operates on the principle of the survival of the fittest. A constant-size population of individuals, each of them is represented by a fixed number of parameters which are coded in binary form (chromosomes), encode possible solutions of a given problem. An initial population of individuals (possible solutions) is generated at random. The allowable range of variation for each parameter is given. There are three main operators that constitute the genetic algorithm search mechanism: selection, crossover and mutation. In every evolutionary step, known as a generation, the individuals of the current population (or family) are decoded and evaluated. Each possible solution is analyzed by a fitness function which decides whether it will contribute to the next generation of solutions. The selection procedure depends on the value of the fitness function. Individuals with high-fitness have a better chance of reproducing, while low-fitness ones will disappear. Once the new population has been selected, chromosomes are ready for crossover and mutation. The crossover operator combines the features of two parents to create new solutions. Crossover allows an improvement in the species in terms of evolution of new solutions at random on each parent and then, complementary fractions from the two parents are linked together to form a new chromosome. The mutation operator alters a copy of a chromosome reintroducing values that might have been lost or creating totally new features. One or more locations are selected on the chromosome and replaced with new randomly generated values.

The three operators are implemented iteratively. Each iteration produces a new population of solutions (generation). The genetic algorithm continues to apply the operators and evolve generations of solutions until a near-optimum solution is found or the maximum number of possible generations is produced. Figure 1 shows the algorithm flow chart.

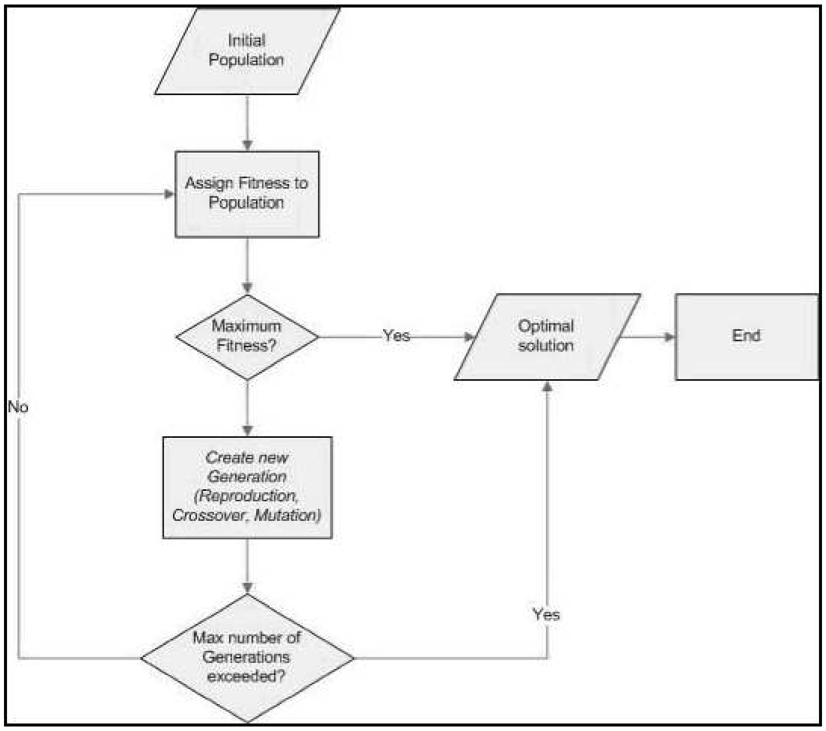

Initial Population
Assign Fitness to Population
Maximum Fitness?
Yes
Optimal solution
End
No
Create new Generation (Reproduction, Crossover, Mutation)
Yes
Max number of Generations exceeded?

Figure 1. The genetic algorithm flow chart.

Note that, differently from classical search methods, the transition rules from one solution to a new solution in the search space are not given in a deterministic form but using probabilistic operators. Besides, differently from the natural case, the size of the new population is kept constant and each new generation is expected to increase the average fitness. This method has been applied by the authors to optimize the trajectory of a fixed wing UAV. In this paper, a control design application for the MicroHawk micro aerial vehicle is discussed. The MicroHawk [6] concept was designed within a European Union funded project (Micro Aerial Vehicles for Multi Purpose Remote Monitoring and Sensing Project), by a research group at Politecnico di Torino. It consists of a fixed wing, tailless integrated wingbody configuration, powered by a DC motor and tractor propeller (see Figure 2). Three versions have been developed and tested, characterized by different size and weight. The reference vehicle - named Mi-

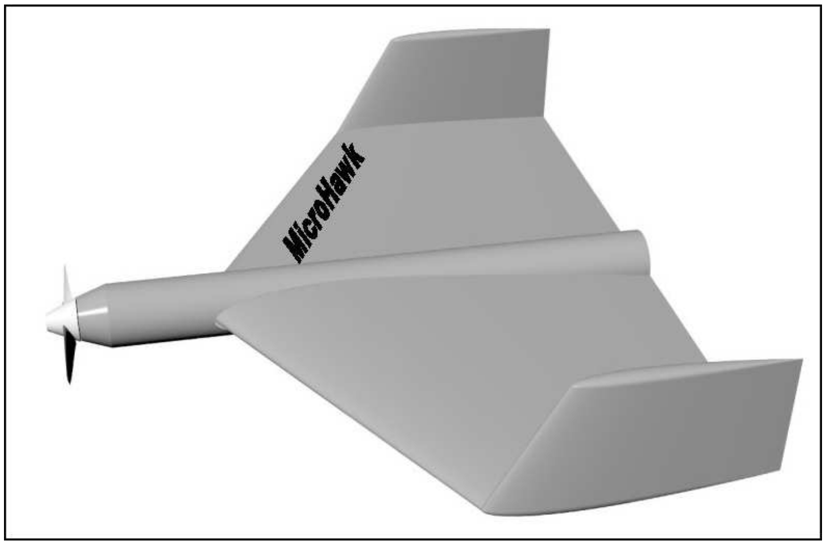


Figure 2. The micro aerial vehicle configuration (MicroHawk).

croHawk600 - is characterized by a 600 mm wingspan and the bare platform weights 400 g. Its design has been mainly adjusted to the need for higher payload weight fraction and larger internal volumes. The MicroHawk600 version can potentially achieve autonomous flight as it is possible to locate onboard a commercial small size autopilot without exceeding wing loading limitations for hand launch.

2. Mathematical Model

2.1 Point-Mass Model

Equations (1) are assumed as the system which describes the dynamics of the vehicle:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{h} \\ \dot{\gamma} \\ \dot{\chi} \\ \dot{V} \end{pmatrix} = \begin{pmatrix} V\cos\gamma\cos\chi \\ V\cos\gamma\sin\chi \\ V\sin\gamma \\ \dfrac{g}{V}\left(n\cos\phi - \cos\gamma\right) \\ \dfrac{g}{V}\dfrac{n\sin\phi}{\cos\gamma} \\ \dfrac{T_e - D}{m} - g\sin\gamma \end{pmatrix}. \tag{1}$$

System (1) is a point-mass model were x, y, h denote the position of the center of gravity (CG) of the aircraft in a ground-based reference frame and are usually referred to as down range (or longitude), cross range (or latitude) and altitude, respectively. Angles are also defined with respect to the same frame: ϕ is the bank angle, χ is the heading angle and γ is the flight-path angle. T_e is the engine thrust, D is the aerodynamic drag, m the aircraft mass, g the gravity acceleration. The ground-speed velocity V is assumed to be equal to the airspeed. The bank angle ϕ, the engine thrust T_e and the load factor $n = \frac{L}{mg}$ are the control variables for the aircraft; hence, we have that the input vector u is:

$$u = [\phi, T_e, n] \,. \tag{2}$$

System (1), complemented with constrains on applicable inputs, form the basis of many aircraft trajectory optimization problems in the literature. Constrains are usually written in terms of original state variables and controls.

The aircraft itself sets some constrains for the state and the control variables like n, T_e, ϕ. As well as constrains on control variables, during the navigation there are limitation on the flight-path angle in both climbing and descending trajectories and on upper and lower bounds of airspeed V, γ. In addition to these constrains, the problem definition includes the initial and terminal conditions for the state variables.

2.2 Cost Function

We want to minimize the following cost function J, which takes in account the error between the nearest point of the trajectory from the desired waypoint and the waypoint itself in terms of position and velocity. Before calculating the cost function a preliminary simulation of the trajectory is done starting from given initial condition. The simulation is performed at equispaced intervals of time. Our interest is focused on finding which step of the simulation is nearest to the objective waypoint. Among the trajectory points, the closest one to the target is used to define the time t_h, which is assumed as the required time to reach the target point in this preliminary trajectory evaluation. The cost function is given in the following form:

$$J = (X_i - X_t)^2 + (Y_i - Y_t)^2 + gk\,(h_i - h_t)^2 + g\,(1-k)\,(V_i - V_{E_{max}})^2 \,, \tag{3}$$

where X_t, Y_t, h_t are referred to as the target waypoint. E_{max} indicates the reference condition which corresponds to the minimum drag condition. The function g is defined as:

$$g = \left(1 - \frac{R}{R_0}\right) , \tag{4}$$

where R_0 is the distance, in the horizontal plane, between the initial waypoint and the target and R indicates, still in the horizonal plane, the distance of the aircraft from the target. k is a weight factor varying from 0 to 1 according to user setup.

2.3 Problem

If we use time discretization and divide time horizon $\hat{t_h}$ over n finite time instants, we obtain a $1 \times n$ vector:

$$\hat{t_h} = [t_0, t_1 = t_0 + \Delta, t_2 = t_0 + 2\Delta, ..., t_0 + n\Delta] \,. \tag{5}$$

Consequently, we have that it is possible to describe the trajectory of an air vehicle as a set of n points at the n time instants, so that it is possible to define a 3 $\times$ n matrix:

$$\{A\} = \left[\begin{pmatrix} \Phi_0 \\ T_0 \\ n_0 \end{pmatrix} \begin{pmatrix} \Phi_1 \\ T_1 \\ n_1 \end{pmatrix} \dots \begin{pmatrix} \Phi_n \\ T_n \\ n_n \end{pmatrix} \right] . \tag{6}$$

We now want to define a simple waypoint distribution and calculate the trajectory including all waypoints which minimizes cost function (Eqn. (3)) all over time horizon $\hat{t_h}$.

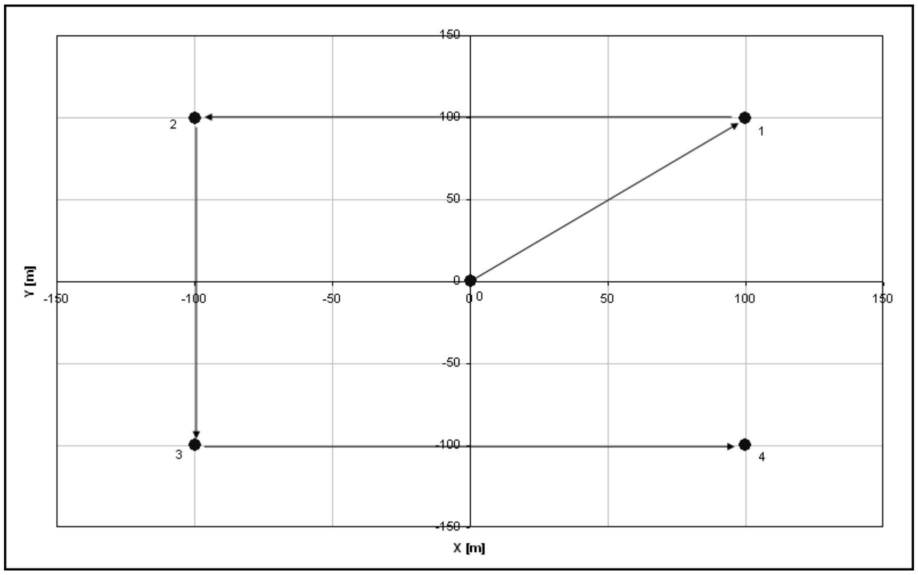


Figure 3. Example of waypoint distribution.

Starting from a $3D$ waypoint distribution as shown in Figure 3 where four waypoints $1, 2, 3, 4$ have been defined in the xy horizontal plane, we will get the $\{A\}_{opt}$ matrix.

2.4 Optimization with Micro-Genetic Algorithm

Elements in matrix A, defined in Eqn. (6), will be the chromosomes of population on which the algorithm will operate in order to minimize cost function

(Eqn. (3)). Figure 4 shows a map of the whole process in which, starting from a generic A matrix, the A_{opt} matrix, satisfying all the constraints, is obtained.

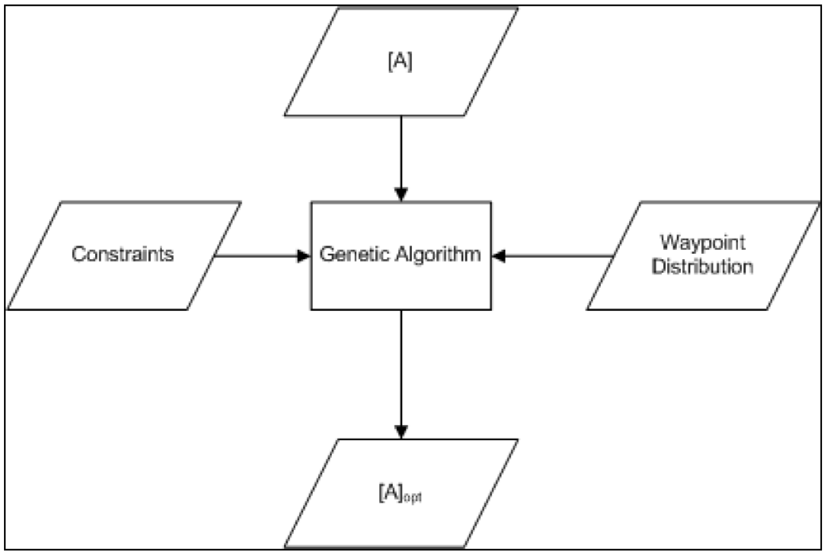


Figure 4. The process map.

The genetic solver adopted for the trajectory optimization is a Fortran version of the driver described by Carroll [3]. The code initializes a random sample of individuals with different parameters to be optimized using the GA approach. The selection scheme used is a tournament selection with a shuffling technique for choosing random pairs for mating. The routine includes binary coding for the individuals, jump mutation, creep mutation and the option for single-point or uniform crossover. Niching, elitism and an option for the number of children per pair of parents are available. Finally, the solution using a micro GA is also possible. This last switch significantly reduced the number of function evaluations and demonstrated faster convergence average to near-optimal region [3, 4]. Note that average population fitness values are not meaningful with a micro-GA because of the start-restart nature of the micro-GA evolution process. Many numerical experiments were performed by Carroll [3, 4] in order to tune the search algorithm adopted and, as a result, the suggested set-up is partially extended for the present application. The code is set for maximum population size of five individuals, 48 bits per individual and three parameters (i.e., 16 binary bits per parameter and 2^{16} possible solutions per parameter). Niching operation is activated. Creep mutation is enabled and one chil per pair of parents is considered. Tables 1 and 2 provide summary of the value of the parameter set in the input file.

Table 1. The input file parameters used in simulation.

Parameter	Value	Parameter	Value
irestrt	0	**icreep**	1
microga	1	**pcreep**	0.04
npopsiz	10	**iunifrm**	1
nparam	3	**iniche**	1
pmutate	0.05	**nchild**	1
maxgen	1,000 $\div$ 100,000	**iskip**	0
idum	-1,000	**iend**	0
pcross	0.5	**nowrite**	1
itourny	1	**kountmx**	1
ielite	1		

Table 2. Other input file parameters.

Parameter	Φ	T_e	n
parmin	-45°	0 N	-1 g
parmax	45°	6 N	2 g
nposbil	32,768	32,768	32,768
nichflg	1	1	1

3. Analysis of the Results

In this section, we simulate a scenario where the transition through four known waypoints is assigned to the UAV platform. The search procedure is initially extended to 1000 generations in order to find the best value of parameter k (Eqn. (3)). Figure 5 shows different trajectories generated by the micro-GA which satisfy, for different values of k, the cost function. In the same figure is represented the trajectory generated by a commercial autopilot set for the MicroHawk vehicle. The dotted circles represent cylinders with a radius of 30 m: we assume that the air vehicle, moving from a target to the other, reaches the destination target when it is inside the cylinder. Figures 6 and 7 show respectively the altitude and the airspeed time-histories for the same previous values of k. For $k = 0.6$ the UAV has the best behaviour: the targets are reached nearest both in the horizontal plane and in altitude range. In this case when the target is reached the altitude difference is less than 5 meters. The velocity time-history shows small velocity decreasing: as a matter of fact we desire to turn as near as possible at V_{Emax} ($< V_{max}$). Moreover, the entire track is concluded faster than in the real flight set-up. It is important to highlight that the micro-GA, differently from the autopilot, aims keeping the maximum speed during first phases of waypoint navigation. Differently, when approaching the waypoint turn, airspeed is reduced to minimum drag (maximum efficiency turn for minimum altitude loss).

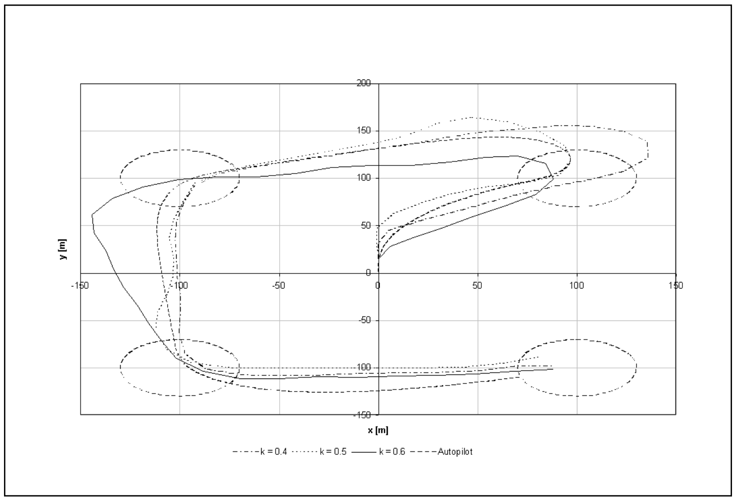


Figure 5. Trajectories for different values of k.

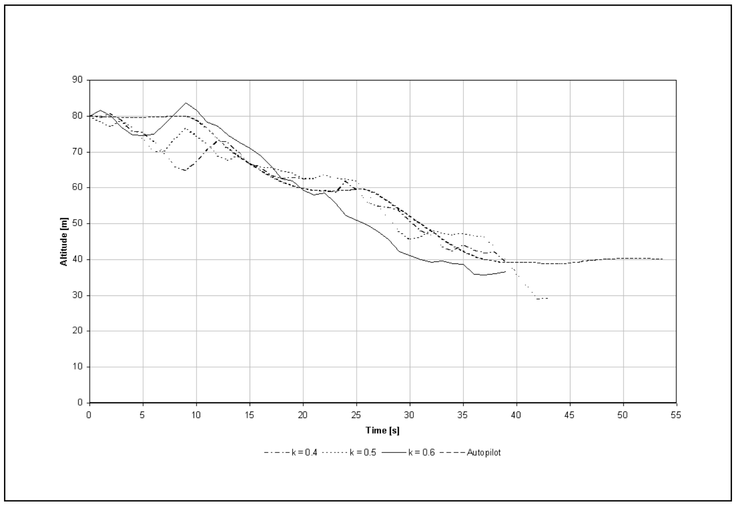


Figure 6. Altitude time-histories for different values of k.

Fixed $k = 0.6$ some additional tests have been carried out increasing the number of generations from 1000 to 100,000. Figures 8, 9, and 10 show the results. The increase of generations still produces an increment of trajectory performance. Unfortunately, the nature of the problem does not allow the

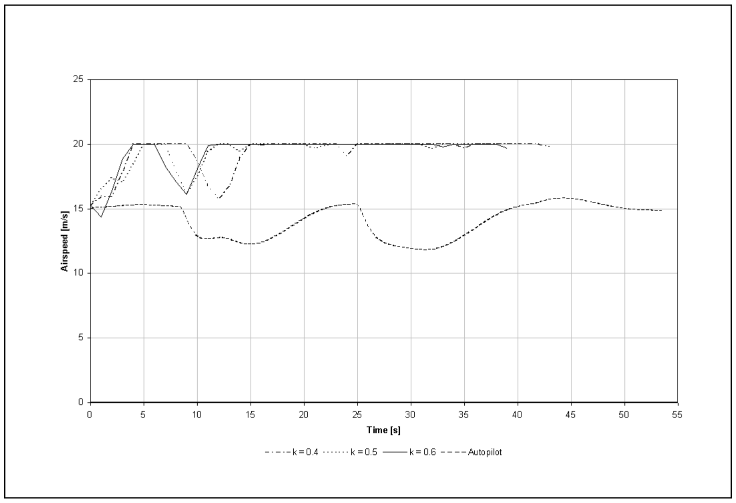


Figure 7. Airspeed time-histories for different values of k.

tracing of the fitness function on the overall flight circuits. Hence, convergence to optimal trajectory can only be decided after flight track inspection.

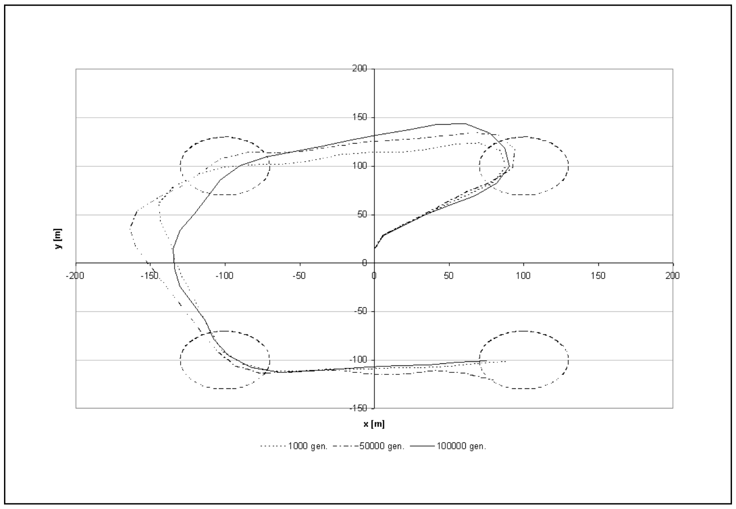


Figure 8. Trajectories for different numbers of generations.

4. Concluding Remarks

The reference trajectory was obtained with a simulator of the real aircraft including autopilot. The model of the controlled system was validated with

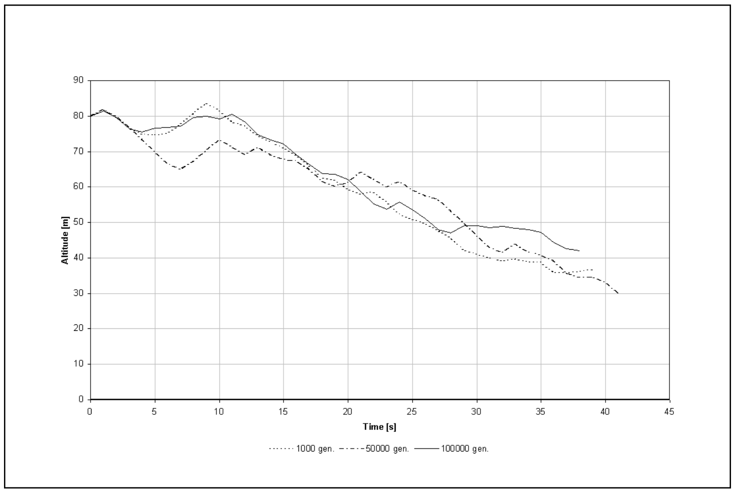


Figure 9. Altitude time-histories for different numbers of generations.

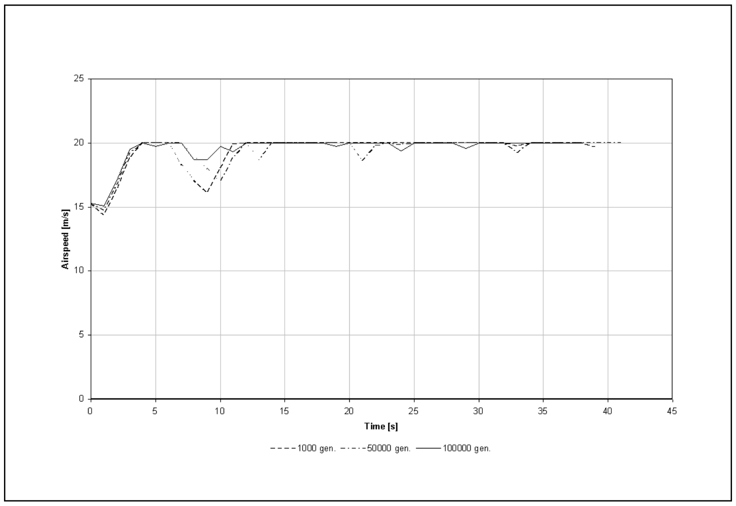


Figure 10. Airspeed time-histories for different numbers of generations.

flight experiments. Gains and setting for the autopilot were set in the simulator in accordance with suggested factory defaults for the application. The purpose of the comparison with optimal trajectories is the upgrade of control and navigation feedback of autonomous system in real flight conditions. Future activity will be devoted to implementation of the optimal search for autopilot gain setting in order to obtain a flexible software tool for appropriate control and navigation, removing the complex trial and error experimental procedure usually suggested by autopilot producers.

References

[1] A. Bemporad. Reference Governor for Constrained Nonlinear Systems. *IEEE Trans. Automatic Control*, 43(3):415–419, 1998.

[2] A. Bemporad, A. Casavola, and E. Mosca. Nonlinear Control of Constrained Linear System via Predictive Reference Management. *IEEE Trans. Automatic Control*, 42(3):340–349, 1997.

[3] D.L. Carroll. *Genetic Algorithms and Optimizing Chemical Oxygen-Iodine Lasers*. Developments in Theoretical and Applied Mechanics, Volume 18, pages 411–424, School of Engineering, The University of Alabama, Tuscaloosa, 1996.

[4] D.L. Carroll. Chemical Laser Modeling With Genetic Algorithms. *AIAA Journal*, 34(2):338–346, 1996.

[5] D.E. Goldberg. *Algorithm in Search, Optimization and Machine Learning*. Addison Wesley, Reading, USA, 1989.

[6] G. Guglieri, B. Pralio, and F. Quagliotti. Design and Performance Analysis of a Micro Aerial Vehicle Concept. In *Proc. 2nd AIAA Unmanned Unlimited Systems, Technologies and Operations Conference*, San Diego, USA, 2003.

[7] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Trans. Robotics and Automation*, 12(4):566–580, 1996.

[8] S.M. LaValle and J.J. Kuffner. Randomized Kinodynamic Planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 473–479, Detroit, USA, 1999.

[9] R. Murray, J. Doyle, J. Marsden, and G. Balas. Robust Nonlinear Control Theory With Application to Aerospace Vehicles. In *Proc. IFAC World Congress*, San Francisco, USA, 1996.

[10] S.J. Qin and T.A. Badgwell. An Overview of Industrial Model Predictive Control Technology. *Chemical Process Control*, 93(316):232–256, 1997.

[11] B. Sweetman. Fighters without Pilots. *Popular Science*, No. 11, pages 97–101, 1997.

[12] TechSat21. `http://www.vs.afrl.af.mil/vsd/techsat21/`.

[13] M. Van Nieuwstadt, M. Rathinam, and R.M. Murray. Differential Flatness and Absolute Equivalence of Nonlinear Control Systems. *J. Control*, 61(6):1327–1361, 1995.

A GENETIC ALGORITHM WITH AN ADAPTATION MECHANISM FOR DATABASE INDEX OPTIMIZATION

Viktor Kovačević
HERMES SoftLab, d. d.
Ljubljana, Slovenia
viktor.kovacevic@hermes.si

Bogdan Filipič
Department of Intelligent Systems
Jožef Stefan Institute, Ljubljana, Slovenia
bogdan.filipic@ijs.si

Abstract Relational database tuning is a complex process which requires various levels of competence, from system and hardware engineering to knowledge of business logic. Optimizing application query workload with selection of proper set of binary indexes that minimize query response time and consecutively the resource usage is known as the index selection problem (ISP). As a stochastic, biologically inspired search method suitable for finding near-optimal solutions in complex search spaces, a genetic algorithm is suitable for solving this problem. In this paper, we present an adaptation mechanism incorporated in the Genetic ALgorithm for Index Optimization (GALIO), an expert tool for ISP. Operational testing of GALIO with the adaptation mechanism on real-world databases shows a significant improvement of optimization results in comparison with the results obtained without adaptation.

Keywords: Database optimization, Genetic algorithm, Index selection problem, Query access path evaluation

1. Introduction

The mainstream in modern database development is the production of relational database management systems with processes and tools based on human knowledge needed for efficient exploitation of database systems. To maintain their data, the users communicate with databases through queries. Database systems need to provide optimal usage of system resources and supply requested data with minimum response time. Binary indexes created on crucial database entity keys and attributes of the most searchable table columns present a fundamental approach to optimization of the query execution cost. Binary indexes themselves are separate database objects that entirely depend on data in the tables. Therefore, they increase the execution cost for data maintenance queries. Finding an optimal index configuration to get the minimal execution cost, balanced between data search and data maintenance queries from the application query workload, is known as the index selection problem (ISP). In real-world relational database systems with hundreds of tables, thousands of table columns and hundreds of different queries, the ISP is a complex combinatorial problem. Furthermore, it is proven that the problem is NP-complete [2].

The index selection problem has been studied since the early seventies and the importance of the problem is widely acknowledged [6, 10]. Most recent releases of database systems, such as Oracle, DB2 and SQL Server, include the so-called index advisors capable of analyzing the workload in terms of costs of previously performed queries, and deriving recommendations for index creation [3, 4, 9]. Various implemented tools as well as the work presented in the literature show that the ISP draws considerable attention of the academic and engineering communities [1, 9, 11]. In modern relational database design, numerous optimization methods and algorithms are being included into application development interfaces and infrastructure.

In our previous work we presented the Genetic ALgorithm for Index Optimization (GALIO) and in preliminary experiments it was shown suitable for the ISP [7]. Furthermore, we have grounded our approach on existing solutions and tools, especially on direct usage of database query cost evaluation methods and application of the optimization algorithm independently from the specific relational database implementation. In this paper we extend the previous research with incorporating an adaptation mechanism into the genetic algorithm. Moreover, we test the algorithm on a real database system.

The paper is further organized as follows. We first describe the problem and summarize the results of the original algorithm. Next, we present the new adaptation module and its impact on the design of genetic operators. Finally, we report the optimization results achieved with the GALIO expert tool on an e-banking system database and conclude with the ideas for future work.

2. Problem Definition

Let $T = \{1, 2, \ldots, n\}$ be a set of tables and $I = \{1, 2, \ldots, m\}$ a set of all combinations of secondary indexes on the tables T (from one indexed column to the predefined number of columns for indexes, Δ_c, usually 3 or 4). Each table contains different number of table columns $C = \{1, 2, \ldots, k_i\}, i = 1, 2, \ldots, n$, where i is the total number of tables in the database system. The probability of column constituent in new index candidates is defined for each column in each table:

$$P_{C_{ij}} = \frac{\mathit{NUM\_DISTINCT}_j}{\sum_{j=1}^{k_i} \mathit{NUM\_DISTINCT}_j},$$

where k_i is the number of columns in table T_i, and $\mathit{NUM\_DISTINCT}_j$ the number of distinct values in column j. Furthermore, we define an index configuration $\Omega = T \bigotimes I$, which denotes that for each table in T we define a subset of secondary indexes $\mathit{SI} \subset I$. For each index configuration we also define the maximum number of indexes per table, Δ_t, usually up to 5. When all indexes of a certain configuration *SI* are built, the query workload cost on that configuration is estimated. The estimation is based on two criteria. The first one is the cost estimated for each query on the index configuration by the database optimizer. The second one is the *SI* maintenance time, which is estimated from the index statistics. For building our prototype tool, only the number of indexed columns is used for the index maintenance factor calculation. The estimation of the total cost for a query workload represents the sum of cost estimations given by the database optimizer and is calculated through a query explanation plan mechanism for each query in the query workload. The explanation plan for the query workload also gives the information about the index access paths as well as the index usage. Therefore, the final index configuration is re-evaluated by cost and by the index usage. At the end, we get a new index configuration $\mathit{SI}' \subset \mathit{SI}$ that contains only the usable indexes and $\mathit{USI} = \mathit{SI} - \mathit{SI}'$ with unusable indexes.

We can use the list of unusable indexes *USI* to change table column statistics for each first column of the unusable indexes so that the new number of distinct values is calculated as

$$\mathit{NUM\_DISTINCT}'_{j_0} = \mathit{NUM\_DISTINCT}_{j_0} \times \beta$$

$$\Leftrightarrow \langle \text{changed} \;\; \mathit{NUM\_DISTINCT} = \lambda \rangle$$

$$\lambda'_{j_0} = \lambda_{j_0} \times \beta.$$

Here β represents the penalty factor with value $0 \leq \beta < 1$. In our work we use the value of the penalty factor between 0 and 0.2.

3. Previous Results with GALIO

In the initial tests of GALIO we achieved encouraging results, especially with regards to efficiency and robustness of the algorithm on a sample production database [7]. Some differences were noted in comparison to human-defined solutions. This particularly holds in case of large tables in terms of the number of records, with some 'usable' columns with relatively small number of distinct values compared to the total number of records in the table. Index candidates on such columns also contribute to a great extent to the total query cost reduction. The initial version of GALIO used static probabilities for column candidates which are based on the number of distinct values from the table data statistics. Therefore, these columns have relatively small probability to be index candidates. This results in not including them in the resulting index configuration. The initial version of GALIO does not change these static probabilities for column candidates during the execution of the algorithm.

Each table in the entity-relationship model has the data statistics structure which contains the parameters used in the process of creating a new index set. The initial version of the algorithm uses a number of different values such as the probability that a column will be used in the index. A column with a higher number of distinct values has a higher probability to be chosen in the new index creation process. The new index creation process is assumed as creation of a new index with predefined number of columns or as addition of a new column to the existing index in order to improve the index selection factor. The index selection factor is defined as the number of matching columns in query filter predicates that are covered by index columns. Previously used columns, either in new index creation or in adding a new column, are not used in the subsequent steps of searching for the next index column. Obviously, each table column can be used in a single index only once.

4. Adaptation Module Design

The improvement of GALIO by adding a new module for adaptation of column index probabilities is based on the information on ‘usability’ of the proposed index configurations in previously evaluated query access paths. The term ‘usability of an index’ is used to specify the usage of the index in the query explain plan access path produced by the database optimizer. During the algorithm execution, probabilities for the columns that are members of ‘unusable’ indexes are reduced, while the probabilities of the other table columns are increased. After a new index configuration is evaluated, that is, query workload access paths are estimated, the algorithm adapts table column probabilities for the next generation. Each time an index is marked as ‘unusable’, the probability of the first index column is decreased. Probabilities of other columns are increased in equal shares of the total probability reduction of unusable columns. This process is carried out after the evaluation of each new index configuration.

The adaptation module uses as its input two groups of parameters. The first group is related to table statistics, while the second group is associated with the index usability and obtained from the usage of previously indexed columns (binary indexes) in the explained query access paths. The main task of the adaptation module is to obtain knowledge from the previously explained queries and to change the probabilities of columns to participate in new index creation. The first group of input parameters is mostly static whereas the second group depends on the algorithm behavior and results. The probabilities of table columns are initialized from table data statistics. The adaptation system changes these probabilities from temporary results of the algorithm in each execution step. This process consecutively directs the search towards the optimal result. Figure 1 shows the integration and the data flow between the adaptation module and the genetic algorithm implemented in the GALIO tool.

5. Modifications of Genetic Operators

In GALIO, each candidate solution (index configuration) is represented in the form of a matrix, where the matrix columns are table columns sorted in the lexical order and grouped by tables in the configuration [7]. The propagation of good genetic material (usable indexes) among index configurations is part of the computer-simulated evolution process. It is performed by two genetic operators: recombination and mutation [5].

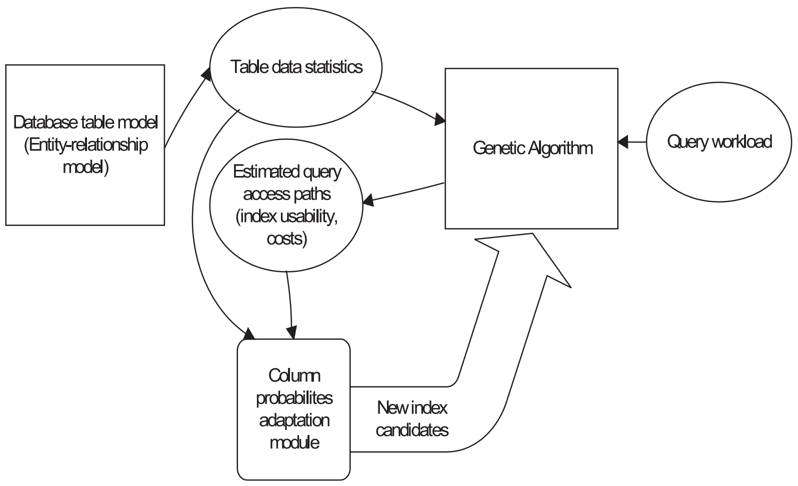


Table data statistics
Database table model (Entity-relationship model)
Genetic Algorithm
Query workload
Estimated query access paths (index usability, costs)
Column probabilites adaptation module
New index candidates

Figure 1. Adaptation module and the genetic algorithm data flow.

Column probabilities based on the number of distinct values are integral part of the index configuration genome. The index configuration matrix includes a row with the number of distinct values for each column in the tables, as shown in Figure 2.

The mutation operator adds a new index to an existing index configuration. After the evaluation of the query work load on the mutated index configuration, the number of distinct values for each first column of unusable indexes recalculated. The applied mutation operator is illustrated in Figure 3.

	T_1C_1	T_1C_2	$\cdots$	T_1C_i	$\cdots$	T_1C_n	$\cdots$	T_n,C_1	T_nC_2	$\cdots$	T_nC_i	$\cdots$	T_nC_n
0	λ_{11}	λ_{12}	$\cdots$	λ_{1i}	$\cdots$	λ_{1n}	$\cdots$	λ_{n1}	λ_{n2}	$\cdots$	λ_{ni}	$\cdots$	λ_{nn}
1	1	2									1		
$\vdots$													
i		1		2		3					2		1
$\vdots$													
n	1			2									

Figure 2. Representation of index configuration in the genetic algorithm.

	T_1C_1	T_1C_2	$\cdots$	T_1C_i	$\cdots$	T_1C_n	$\cdots$	T_n,C_1	T_nC_2	$\cdots$	T_nC_i	$\cdots$	T_nC_n
0	λ_{11}	λ_{12}	$\cdots$	λ_{1i}	$\cdots$	λ_{1n}	$\cdots$	λ_{n1}	λ_{n2}	$\cdots$	λ_{ni}	$\cdots$	λ_{nn}
1	1	2									1		
$\vdots$													
i		1		2		3					2		1
$\vdots$													
n	1			2									
$n+1$				~~1~~	— — —	~~2~~							
$n+2$									~~1~~	— — —	~~3~~	— — —	~~2~~

$\lambda'_{1i} = \lambda_{1i} \times \beta$ $\lambda'_{n2} = \lambda_{n2} \times \beta$

Figure 3. The adapted mutation operator.

The recombination operator is more complex. First, the parents simply add their index configurations, like in the original recombination operator in GALIO. The default row is recalculated in a way that minimal values for the number of distinct values from parents are inherited in the offspring configuration:

$$\forall i,j\ \ \gamma_{ij}^{\textit{offspring}} = \min\langle\lambda_{ij}^{\textit{first}}, \lambda_{ij}^{\textit{second}}\rangle|^{i \text{ table index, } j \text{ column index}}$$

The numbers of distinct values calculated in this way represent a new row in the offspring configuration matrix. After the query workload is evaluated and the usability of indexes is known, the number of distinct values for each first column of unusable indexes is recalculated again. This process is illustrated in Figure 4.

6. Test Environment and Results

The test database system contains a copy of the real production data from an e-banking application for the last four years. The query workload is based on a two-week application log and covers all implemented application query func-

	T_1C_1	T_1C_2	$\cdots$	T_1C_i	$\cdots$	T_1C_n	$\cdots$	T_n,C_1	T_nC_2	$\cdots$	T_nC_i	$\cdots$	T_nC_n
0	λ_{11}	λ_{12}	$\cdots$	λ_{1i}	$\cdots$	λ_{1n}	$\cdots$	λ_{n1}	λ_{n2}	$\cdots$	λ_{ni}	$\cdots$	λ_{nn}
1	1	2									1		
$\vdots$													
i		1		2		3					2		1
$\vdots$													
n	1			2									

$\times$

	T_1C_1	T_1C_2	$\cdots$	T_1C_i	$\cdots$	T_1C_n	$\cdots$	T_n,C_1	T_nC_2	$\cdots$	T_nC_i	$\cdots$	T_nC_n
0	λ_{11}	λ_{12}	$\cdots$	λ_{1i}	$\cdots$	λ_{1n}	$\cdots$	λ_{n1}	λ_{n2}	$\cdots$	λ_{ni}	$\cdots$	λ_{nn}
1	1	2									1		
$\vdots$													
i		1		2					1		2		
$\vdots$													
j				1									

$=$ $\forall i,j\ \gamma_{ij}^{\textit{offspring}} = \min\langle\lambda_{ij}^{\textit{first}}, \lambda_{ij}^{\textit{second}}\rangle|^{i\text{ table index},\ j\text{ column index}}$

	T_1C_1	T_1C_2	$\cdots$	T_1C_i	$\cdots$	T_1C_n	$\cdots$	T_n,C_1	T_nC_2	$\cdots$	T_nC_i	$\cdots$	T_nC_n
0	λ_{11}	λ_{12}	$\cdots$	λ_{1i}	$\cdots$	λ_{1n}	$\cdots$	λ_{n1}	λ_{n2}	$\cdots$	λ_{ni}	$\cdots$	λ_{nn}
1	~~1~~	2									1		
$\vdots$													
i		1		2		3					2		1
$\vdots$													
n	1			2									
n+1	1	2									1		
$\vdots$													
$n+i$		~~1~~		~~2~~					~~1~~		~~2~~		
$\vdots$													
$n+j$				1									
	$\gamma'_{11} = \gamma_{11} \times \beta$	$\gamma'_{12} = \gamma_{12} \times \beta$							$\gamma'_{n2} = \gamma_{n2} \times \beta$				

Figure 4. The modified recombination operator.

tionalities. The query log contains 84 different types of queries from regular usage. The total number of query executions is 620,716. For individual queries the number of executions varies from 2 to 52,505. The query complexity increases from the most simple (one table select queries or updates) to highly complex queries with 10 tables, with sub-queries and unions. The percentage of different query types in entire query workload is: 59 % of select queries, 28 % of update queries, 11 % of insert queries, and 2 % of delete queries. The query complexity is expressed as the number of elementary database operations needed to obtain the query result records from the tables. The most frequently used operations are full scans, index scans, different joins and nested loops. On average, each query in the workload has 8.46 different elementary database operations. The physical size of the database files is 13 GB. The total number of columns for tables in the model (882) is constrained to the number of 198 different table columns favorable for secondary indexes.

The imported query workload statistics influence the genetic algorithm settings. The most important factor is the average number of columns in query search criteria for table records. For the testing query workload, it was the average number of 2 to 3 different column search criteria per table. GALIO was configured to search for optimum secondary indexes of up to 3 columns. Another crucial factor is the number of indexes per table. This configuration parameter is set to the maximum of 5 indexes on each table. An advantageous property of GALIO is removing of indexes not used in a query explain paths from the proposed index configuration. This allows setting up any value for the maximum number of indexes per table higher than the lower limit of the number of the indexes used in query explain paths. Large value of this parameter can cause unnecessary combinatorial complexity of the search and low performance of the genetic algorithm.

Mutation and recombination probabilities were set to balance with equal probability between the two genetic operators. As a result, on average half of individuals were mutated and recombined in each generation, without exclusive right to one or other operator for a specific individual. It is also possible that an unchanged individual passes to the next generation. The tested population sizes were 10 to 30 individuals, and the population of 15 individuals was found a compromise between time efficiency and resource requirements of the applied algorithm. The average execution time was up to 30 minutes on a nowadays standard Pentium hardware configuration where the database server and the genetic algorithm ran on the same machine.

A typical resulting index configuration contains 38 different secondary indexes. There are two tables with 4 indexes recommended (payments and accounts tables with high employability in the e-banking application system). We also have two tables with 3 indexes (payment packages and user company relation table). The remaining indexed tables have one index (30 % of total number of 44 tables) and two indexes (30 % of total number of tables). These values demonstrate suitable algorithm behavior in deleting unusable indexes from index configurations. There are 17 tables (38not indexed. They, without exception, belong to a group of tables with small number of records. On average, one-column indexes contribute with approximately 60 %, two column indexes with 30 %, and three column indexes with 10 % to the total number of indexes in the resulting configuration.

The best result of the original GALIO algorithm was compared with the result of the modified algorithm and with the human-defined index configuration. The resulting index configurations were evaluated with the absolute value of cost function from the original algorithm [7]. The result of the modified algorithm (cost value 3,409) is 34 % better than the query workload cost result of the human-defined secondary index configuration (cost value 5,659). Moreover, in comparing the original and modified algorithm, the result is even more significant. The index configuration cost for the modified algorithm represents only 40 % of the original GALIO result (cost value 8,509).

7. Conclusion

After the described phase of development, the genetic algorithm for the database index selection achieves significant results in real-world e-banking database application, especially with respect to efficiency and robustness. Satisfactory results are also achieved in comparison with human-defined solutions. Future work will be concerned with improving the algorithm and comparing its results with those of other algorithms for the index selection problem as well as with the results of database index advisors included in commercial database management systems. We also plan to improve the evaluation method for index maintenance cost and extend it to include other index statistics information. Furthermore, it is possible to incorporate index data file space configuration parameters and other types of indexes, like bit-mapped or functional indexes, into the existing genetic algorithm. It seems is possible to design a general biologically inspired algorithm for tuning database management systems with various data structures, not only b-tree indexes, for query optimization.

References

[1] S. Chauduri and V.R. Narasayya. An efficient cost-driven index selection tool for Microsoft SQL server. In *Proc. 23rd International Conference on Very Large Data Bases*, pages 146–155, Athens, Greece, 1997.

[2] D. Corner. The difficulty of optimum index selection. *ACM Trans. Database Syst.*, 3(4):440–445, 1978.

[3] Database Engine Tuning Advisor Reference. `msdn2.microsoft.com/en-us/library/ms173494.aspx`.

[4] DB2 Advisor: An optimizer smart enough to recommend its own indexes. `www-128.ibm.com/developerworks/db2/library/techarticle/dm-0107lohman/`.

[5] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin, Heildelberg, New York, 2003.

[6] M.M. Hammer and A.Y. Chan. Index selection in a self adaptive database management system. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 1–8, Washington, DC, USA, 1976.

[7] V. Kovačević and B. Filipič. A genetic algorithm based tool for the database index selection problem. In *Proc. 8th International Multiconference Information Society (IS 2005), Intelligent systems*, pages 378–381, Ljubljana, Slovenia, 2006.

[8] Oracle Using Advisors to Optimize Database Performance. `download-east.oracle.com/docs/cd/B16240_01/doc/server.102/b14196/montune003.htm`.

[9] K. Sattler, I. Geist, and E. Schallehn. QUIET: Continuous query-driven index tuning. in *Proc. 29th International Conference on Very Large Data Bases*, pages 1129–1132, Berlin, Germany, 2003.

[10] M. Schkolnik. The optimal selection of secondary indices for files. *Inf. Syst.*, 1(4):141–146, 1975.

[11] G. Valentin, M. Zuliani, D. Zilio, G. Lohman, and A. Skelley. DB2 Advisor: An optimizer smart enough to recommend its own indexes. In *Proc. 16th International Conference on Data Engineering (ICDE 2000)*, pages 101–110, San Diego, CA, USA, 2000.

AN EXPERIMENTAL STUDY ON GA REPLACEMENT OPERATORS FOR SCHEDULING ON GRIDS

Fatos Xhafa
Department of Languages and Informatics Systems
Technical University of Catalonia, Barcelona, Spain
fatos@lsi.upc.edu

Abstract Computational Grids (CG) represent new computational frameworks that offer large computational power by connecting geographically distributed resources. Obtaining efficient and optimal assignments of jobs to the grid nodes is a main issues in such distributed environments. In this paper, we present a basic GA for Scheduling Jobs on Computational Grids and study two versions of it based on the replacement operators: Steady-State Genetic Algorithm (SSGA) and Struggle Genetic Algorithm (SGA). Considering the value of makespan, we aim to compare their behavior in a real CG. The interest of SSGA is its accentuated convergence of the population that rapidly reaches good solutions although it is soon stagnated. The SGA is based on a struggle replacement policy that adaptively maintains diversity over population. The experimental results show that SGA outperforms SSGA for moderate size instances. On the other hand, for larger size instances, SGA is not able to improve the results obtained by the SSGA.

Keywords: Computational grids, Genetic algorithms, Replacement operators, Scheduling

1. Introduction

The constant growth of communications, in terms of quality and availability, is increasing the interest on grid computing paradigm [5] by which geographically distributed computing resources can be logically coupled together working as a computational unit. An efficient use of distributed resources is highly dependent on the resource allocation by grid schedulers. Moreover, due to the dynamics of a CG, grid schedulers must generate optimal schedules at a minimal amount of time. Job Scheduling on Computational Grids is multiobjective: makespan, flowtime and resource utilization are among most important criteria. In this work, makespan and flowtime are both optimized, but only makespan is reported and used for comparing purpose.

Several heuristics are being addressed in the literature for Job Scheduling on Computational Grids [1, 4, 8, 11]. In particular, Genetic Algorithms (GA) [7] have proved to be a good alternative for solving combinatorial optimization problems. One important characteristic of GAs is the tendency of the population to converge to a fixed point where all individuals share almost the same genetic characteristics. If this convergence is accelerated, by means of the selection and replacement strategy, good solutions will be faster obtained but, population will rapidly converge to worse solutions than those that could have been found if a slower convergence had been maintained. Thus, an appropriate balance of selection pressure must be used to increase the quality of solutions.

In this paper, we present a basic GA algorithm for the problem, and an experimental study on two replacement operators: steady-state and struggle replacement. First, we consider the steady-state replacement strategy, in which only a portion of the population, the worst individuals, is replaced by the newly generated ones. Thus, the selection pressure is increased and as a result the population converges prematurely to a sub-optimal solution. By this manner, the quality of solutions is rapidly increased although the algorithm is soon stagnated. Then, we consider an implementation of a Struggle Genetic Algorithm (SGA), where a new individual replaces the individual that is most similar to it rather than replacing the worst one. The SGA is similar to the Steady-State Genetic Algorithm (SSGA) but it is able to adaptively maintain diversity among individuals, thus aspiring to better solutions.

Several grid scenarios have been considered to study the behavior of the replacement operators. The experimental results show that SGA performs better than SSGA for moderate grid sizes, but as the grid size increases SGA is not able to reach as good results as those of SSGA. More precisely, SGA improves makespan values obtained by SSGA maintaining a similar convergence for small size instances presented in Braun et al. [3] that are currently used as a benchmark for the problem. However, for larger size instances, SGA maintains a too diversified population, which prevents it from improving makespan values obtained by SSGA. This shows how good an intensive policy performs when the grid scenario gets larger, as compared to an explorative policy, especially in a real time environment.

The rest of the paper is organized as follows. In Section 2 we give the problem definition and in Section 3 the basic GA is detailed. The Steady-State and Struggle replacement operators are explained in Section 4. In Section 5 we present the experimental results of the SGA and the SSGA and compare their behavior. Finally, we conclude in Section 6 with some remarks and indicate directions for future work.

2. Problem Definition

Job Scheduling on Computational Grids consists of a dynamic set of independent jobs to be scheduled on a dynamic set of resources. An instance of the problem, at a certain instant of time, is characterized by:

- A set of N independent jobs to be scheduled. Each job has associated its workload (in million of instructions). Every job must be entirely executed in unique machine.
- A set of M heterogeneous machines with ready time value for each machine indicating when this machine is available. Each machine has also associated its corresponding computing capacity (in *mips*).
- An $N \times M$ matrix ETC (Expected Time to Compute) ($ETC[i][j]$ is the expected execution time of job i in machine j.)

Regarding the optimization criteria, makespan and flowtime are both minimized. The makespan of a schedule consists of the completion time of the last processed job; the flowtime consists of the sum of the completion times of each job in the schedule. By letting c_j the completion time job j finishes processing, the two objectives are formally defined as:

$$\textit{makespan}: \min_{S_i \in Sched} \{ \max_{j \in Jobs} c_j \} \text{ and } \textit{flowtime}: \min_{S_i \in Sched} \{ \sum_{j \in Jobs} c_j \},$$

where $Sched$ is the set of all possible schedules and $Jobs$ the set of all jobs to be scheduled. Notice that both objectives are contradictory.

3. GAs for Scheduling on Computational Grids

The starting point for this work was the development of a basic GA implementation for the problem using an adaptation of GA skeleton presented in [2]. The genetic representation, the specific optimization criteria as well as the genetic operators used are described next.

Genetic representation. Each individual encodes a solution by means of a vector containing the schedule. Each position of the vector represents a job and its value indicates the machine it is assigned to. Vectors have size of N and their values are positive integers in $[1, M]$. Thus, all possible representations are feasible solutions. Incompatibilities between jobs and machines have not been considered as infeasibility in this work, but they can be represented adding a penalization in the ETC matrix for the corresponding job and machine (a value of $+\infty$).

Evaluation. Every individual has a fitness value used to measure the quality of solution it represents. Values of makespan and flowtime are both minimized although only makespan is reported here. The approach adopted here is the simultaneous one by which fitness function considers both values simultaneously. We have to take into account that even though makespan and flowtime are measured in the same unit (seconds), the values they can take are in incomparable ranges, due to the fact that flowtime has a higher magnitude order over makespan, and its difference increases as more jobs and machines are considered. For this reason, the value of mean flowtime, $flowtime/N$, is used to evaluate flowtime. Additionally, both values are weighted in order to balance their importance. Fitness value is thus calculated as: $fitness = \lambda \cdot makespan + (1 - \lambda) \cdot mean\_flowtime$, where λ will *a priori* be fixed.

Population initialization. The individuals of the population are randomly generated to create the first generation. Additionally, one individual is generated using the *Longest Job to Fastest Resource - Shortest Job to Fastest Resource* (LJFR-SJFR) heuristic given in [1], which optimizes alternatively both values of makespan and flowtime. A third method used for initialization is the *Minimum Completion Time* heuristic (MCT), described in [3], which computes a possible solution by allocating each job to the machine in which it will finish earlier.

Selection operator. For each generation, an intermediate population is formed by selecting pairs of individuals from the global population to produce the offspring. The selection strategy is a key factor to control selection pressure during the evolution. We have used the k-*Tournament*: for each individual to be selected for the intermediate population, k individuals are randomly chosen from the global population and the best fitted of them is copied onto the intermediate population.

Crossover operator. We have used the *Fitness Based Crossover* by which the crossing mask is built according to the fitness of the two solutions to be crossed.

Mutation. In this implementation we used the rebalance mutation operator, which tries to reduce the workload of one of the most overloaded machines (in terms of their completion times) by swapping if possible or moving jobs from the overloaded machine. After the rebalancing is done, the solution is mutated by applying the Move mutation that randomly moves jobs from one machine to another one.

4. Replacement Operators

The main focus of this work is on two replacements operators, namely steady-state and struggle replacement. In both cases, the algorithm works with an overlapping population where in each generation a portion of the population is replaced by the new individuals, maintaining the size of the population constant. In this way, there is another inherent selection mechanism that rejects the portion of the population to be replaced through which the selection pressure can be easily regulated.

4.1 Steady-State Genetic Algorithm (SSGA)

The steady-state strategy was popularized by the GENITOR program [10]. It consists of the replacement of the worst individuals by the newly generated ones. Consequently, the best individuals are considerably favored and the population often converges prematurely. However, although there is risk of stagnation, SSGA performs very well if good solutions have to be rapidly found. This is the case of Scheduling on Grids where resource allocation is constrained by a time limit.

4.2 Struggle Genetic Algorithm (SGA)

The Struggle GA developed in [6] is similar to SSGA. However, in the SGA, a new individual replaces the individual that is most similar to it only in case the new individual obtains a better fitness value than the one to be replaced. This is done in order to adaptively maintain certain diversity among the population and thus aspiring to better solutions. In order to compare the similarity between solutions, a measure of similarity or distance function has to be defined. In our case, we have used Hamming distance to evaluate similarity between two solutions.

Another issue to be considered is that struggle replacement is strongly constrained by its computational cost of quadratic order of population size. In order to obtain a linear cost, we have designed a hash table to find, given a newly created individual, the individual most similar to it in a constant computational cost.

5. Experimental Study

We conducted an experimental study, initially to tune the parameters of the basic GA, obtain computational results for SSGA and SGA algorithms and compare the behavior of the two replacement operators.

Instance description. The instances used for the experimenting consists of, on the one hand, (a subset of) instances given in Braun et al. [3], known for its high level of difficulty. All these instances consist of 512 jobs and 16 machines. We will refer to these instances as *benchmark*. Further, because the benchmark instances are of rather small size, we have generated a set of instances of larger sizes following the ETC matrix model of Braun et al. [3].

We have generated instances of four different sizes (Small, Medium, Large and Very Large) according to the number of jobs and machines, as shown in Table 1. These instances consists of inconsistent ETC matrices with high task heterogeneity and high machine heterogeneity.

Table 1. Sizes of static instances.

	Benchmark	Small	Medium	Large	Very Large
No. Jobs	512	512	1024	2048	4,096
No. Machines	16	32	64	128	256

Fine tuning of parameters and operators. All the parameters of the GA implementation have been set up in order to obtain the best behavior of SSGA and SGA; the resulting configuration is then used for both them for the rest of the experimental study. Regarding optimization criteria, more priority is given to makespan over mean flowtime ($\lambda = 0.75$). Population size has been set according to instance size; intermediate population size corresponds approximately to 60 % of population size (see Table 2 for specific values).

Table 2. Population sizes.

	Benchmark	Small	Medium	Large	Very Large
Pop. Size	10	35	40	45	50
Int. Pop. Size	6	20	24	26	30

The rest of the parameters are set as follows: *mutate probability* $= 0.4$ and k-*Tournament parameter* $= 3$. The search has been limited to 90 seconds, which is commonly used as a reasonable amount of time for scheduling jobs in a Computational Grid environment (see also [3]).

Computational results using benchmark instances. Instances from [3] were very useful to get a first evaluation of our implementation. The experimental results for this set of instances (see Table 3) are obtained on an AMD K6^{TM} 3D 450 MHz processor with 256 MB of RAM. Results are averaged over 10 runs. We give in Table 3 also the results obtained by the GA implemented in [3] for the same instances. It is worth to note that the implementation of the Braun et al. uses a population of 200 individuals and the heuristic of *Min-Min* to initialize it. Their executions were done on a Pentium II 400 MHz processor with 1 GB of RAM using, in average, an execution time of 90 seconds.

Table 3. Results obtained for benchmark instances (the notation u_x_yyzz.0 means: u–uniform distribution, x–inconsistency (c–consistent, i–inconsistent and s–semi-consistent), yy–job heterogeneity (hi–high, lo–low), zz–machine heterogeneity (hi–high, lo–low)).

Instance	Braun et al. GA	SSGA	SGA
u_c_hihi.0	8,050,844.5	*7,766,109.88*	**7,752,689.08**
u_c_hilo.0	156,249.2	*156,032.18*	156,680.58
u_c_lohi.0	258,756.77	*251,621.13*	**253,926.06**
u_c_lolo.0	5,272.25	*5,242.02*	**5,251.15**
u_i_hihi.0	3,104,762.5	3,216,911.63	3161,104.92
u_i_hilo.0	75,816.13	76704.43	**75,598.48**
u_i_lohi.0	107,500.72	113,972.01	111,792.17
u_i_lolo.0	2,614.39	2,667.73	2,620,72
u_s_hihi.0	4,566,206.00	*4,509,660.58*	**4,433,792.28**
u_s_hilo.0	98,519.4	99,859.48	98,560.04
u_s_lohi.0	130,616.53	131,796.29	**130,425.85**
u_s_lolo.0	3,583.44	3,600.79	**3,534.31**

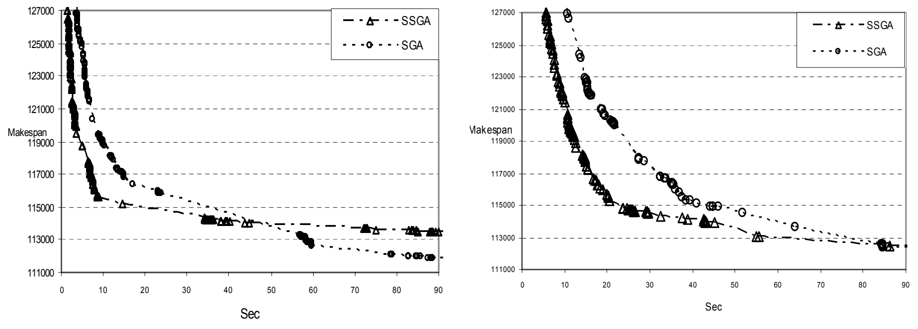


Figure 1. Makespan reduction by SSGA and SGA for the instance u_i_lohi.0 using a population of 10 individuals (left) and 30 individuals (right).

As can be seen from Table 3, the results obtained by SSGA and SGA reach the same level of quality as those obtained by the implementation from [3] taken as reference in our work. SSGA outperforms results of [3] for almost half of the instances having an average of deviation of 2.23 % from the best known value for the rest of instances (6.02 % in the worst case). SGA outperforms more than half of the results obtained by the reference GA having a deviation of 1.27 % in average for the other instances (3.99 % in the worst case). This shows that good results can be obtained despite the selective policy followed by SSGA and SGA, which force a fast convergence in order to reach a fast reduction of makespan.

On the other hand, the results also show that SGA outperforms SSGA for the majority of instances, mainly for inconsistent and partially consistent matrices (this means that SGA performs better when job-machine constraints have to be managed). Figure 1 shows makespan reduction obtained by the SSGA and SGA. Both algorithms perform an accentuated reduction in time rapidly reaching good values, however SGA maintains more diversity among population thus reducing its tendency to converge and reaching better results than those of SSGA.

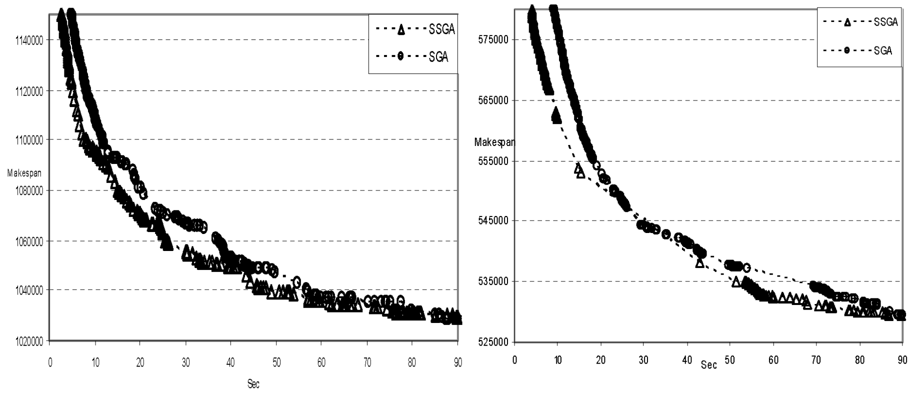


Figure 2. Makespan reduction by SSGA and SGA for the small instance scenario (left) and medium instance scenario (right).

The experiments show a certain constancy on the number of iterations needed by SGA to reach SSGA results. However, as the population size is increased, the point where SGA improves SSGA is delayed (see Figure 1). This is explained by the fact that now the computation time of each iteration is higher due to the larger population.

Computational results using larger size instances. Larger size instances have been generated in order to represent scheduling environments of a larger magnitude. This time, the executions are done in an AMD Athlon^{TM} XP 1600+ (1400 MHz) processor with 256 MB of RAM. Again, the search has been limited to 90 seconds. We show in Table 4 the results (averaged over 10 runs) for the makespan for different grid scenarios: small, medium, large and very large, respectively.

From Table 4 we can clearly observe the diminution of the advantage that SGA obtains over SSGA when the instance is larger (see also Figures 2 and 3.) SGA obtains a better balance between exploitation and exploration of search space reaching better results than those of SSGA. However, for a real grid where the number of jobs and machines is highly variable, SSGA gives a higher guarantee of a satisfactory performance.

Table 4. Results obtained for larger static instances.

Instance	SSGA	SGA
Small	1,029,592.60	**1,027,901.58**
Medium	529,425.13	**529,365.42**
Large	**282,460.00**	286,614.78
Very Large	**160,993.02**	168,804.68

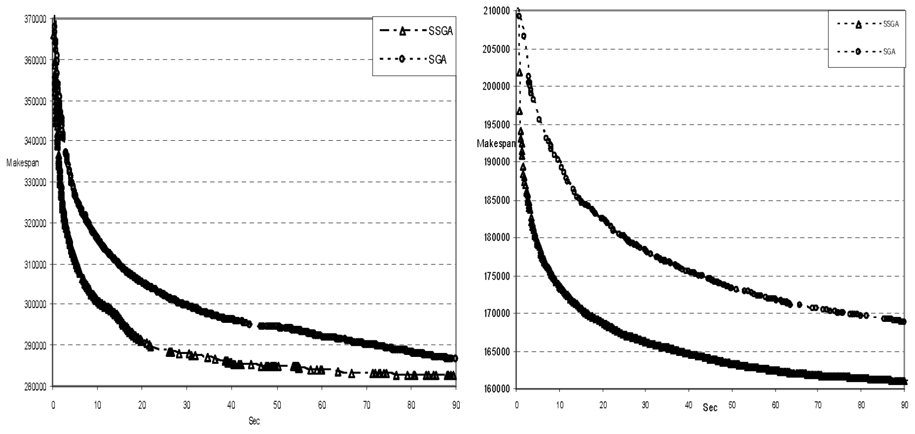


Figure 3. Makespan reduction by SSGA and SGA for the large instance scenario (left) and very large instance scenario (right).

6. Conclusions and Further Work

In order to exploit the potential of a Computational Grid, any grid scheduler must provide good schedules in a reasonable amount of time. We have studied two known versions of GA heuristic for the scheduling problem, namely Steady-State GA (SSGA) and Struggle GA (SGA). The results of this work show that, even though SGA outperforms SSGA when considering a moderate number of jobs and machines, for larger scenarios SGA maintains too high diversity and it is not able to reach the results obtained by SSGA. Moreover, as more jobs and machines are considered, the distance between the makespan reduction obtained by SSGA and SGA is rapidly increased and thus making SSGA more adequate for dynamic grid environments.

Although different grid scenarios have been used, our ultimate goal is to study the performance of SGA and SSGA on dynamic environment. We are currently testing a grid simulator based on HyperSim package [9] that we will use to study the performance of SSGA and SGA.

Acknowledgement

This research is supported by Projects ASCE TIN2005-09198-C02-02 and FP6-2004-IST-FETPI (AEOLUS).

References

[1] A. Abraham, R. Buyya, and B. Nath. Nature's heuristics for scheduling jobs on computational grids. In *Proc. 8th IEEE International Conference on Advanced Computing and Communications*, Cochin, India, 2000.

[2] E. Alba, F. Almeida, M. Blesa, J. Cabeza, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, C. León, J. Luna, L. Moreno, C. Pablos, J. Petit, A. Rojas, and F. Xhafa. MALLBA: A library of skeletons for combinatorial optimisation. *Lect. Notes Comput. Sc.*, 2400:927–932, 2002.

[3] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, and B. Yao. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel and Distr. Comput.*, 61(6):810–837, 2001.

[4] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In *Proc. 4th International Conference on High Performance Computing in Asia-Pacific Region*, Beijing, China, 2000.

[5] I. Foster and C. Kesselman. *The Grid - Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[6] T. Gruninger. Multimodal optimization using genetic algorithms. Master's thesis, Stuttgart University, 1996.

[7] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[8] V. Di Martino and M. Mililotti. Sub optimal scheduling in a grid using genetic algorithms. *Parallel Comput.*, 30(5-6):553—565, 2004.

[9] S. Phatanapherom and V. Kachitvichyanukul. Fast simulation model for grid scheduling using hypersim. In *Proc. 35th Winter Simulation Conference*, New Orleans, LA, USA, 2003.

[10] D. Whitley. The genitor algorithm and selective pressure: Why rank-based allocation of reproductive trials is best. In *Proc. 3rd International Conference on Genetic Algorithms*, pages 116–121, Fairfax, VA, USA, 1989.

[11] A.Y. Zomaya and Y.H. Teh. Observations on using genetic algorithms for dynamic load-balancing. *IEEE Trans. Parallel and Distributed Systems*, 12(9):899–911, 2001.

A GRID-BASED PARALLEL OPTIMIZATION ALGORITHM APPLIED TO A PROBLEM IN METAL CASTING INDUSTRY

Jürgen Jakumeit
ACCESS e. V.
Aachen, Germany
jakumeit@access.rwth-aachen.de

Thomas Barth, Julian Reichwald, Manfred Grauer, Frank Thilo
Information Systems Institute
University of Siegen, Germany
{barth,reichwald,grauer,thilo}@fb5.uni-siegen.de

Thomas Friese, Matthew Smith, Bernd Freisleben
Department of Mathematics and Computer Science
University of Marburg, Germany
{friese,matthew,freisleb}@informatik.uni-marburg.de

Abstract Since customers' quality requirements in casting industry are constantly increasing while 'time to market' must be reduced at the same time, utilizing numerical simulation of the physical casting process and its subsequent optimization is an important topic in casting industry. Simulation-based optimization of casting processes requires consideration of these characteristics: Handling the numerical properties of the optimization problem and the demand for computational resources due to excessive runtime of simulation. In this paper, a distributed optimization algorithm is presented, integrating features from 'traditional', deterministic optimization algorithms, their parallel extensions and Genetic Algorithms. In order to answer the demand for computational resources, its implementation within a Grid Computing infrastructure is briefly discussed. Similarities between the aforementioned classes of algorithms allow their application within the same Grid-based environment. Preliminary results from such an environment integrating optimization algorithm and a simulation code for metal casting are also presented. The developed framework and the application show that grid technology can be an important tool to utilize a variety of optimization techniques and the necessary resources for the optimization of industrial processes.

Keywords: Distributed optimization, Grid computing, Metal casting processes, Numerical simulation

1. Introduction

The design of an optimization algorithm appropriate to solve 'real life' problems in an engineering domain like metal casting has to take several aspects into account simultaneously:

- **Mathematical characteristics of the optimization problem** Properties of the search space such as multimodality, ruggedness, non-differentiable objective and/or constraint functions imply the use of direct methods. Since finding the (a priori unknown) global optimum in such a search space is almost impossible to assure (e.g. by a mathematical proof) for general problems, it is widely accepted to apply heuristics to approximate the global optimum.

- **Characteristics of the solution process** Due to the fact that objective and/or constraint functions are given implicitly by computationally expensive numerical simulation codes a 'traditional' sequential solution process is inapplicable. In order to get a solution to a problem in reasonable time non-sequential optimization algorithms are a mean to speed up the solution process substantially.

- **(Software-)Technical requirements on an adequate problem solving environment** High demand for computational resources necessary to solve typical problems exhibiting the aforementioned characteristics leads to non-traditional approaches to the design and implementation of distributed problem solving environments.

Service-oriented Grid computing has gained tremendous interest in various application domains. Many of those applications stem from an academic environment and have traditionally been designed as monolithic solutions that are hard to adapt, even to slight changes in the application requirements. Required adaptations must be implemented by programmers specialized both in Grid middleware and the applications. The paradigm shift to service-orientation in Grid middleware opens the possibility to use a far more flexible software development approach, namely to compose applications from standard components, promising easier development and modification of Grid applications. Even though, Grid technology has only seen a slow adoption in commercial application domains such as engineering. We see two main reasons for this slow adoption: On the one hand, the inherent complexity of current service-oriented

Grid middleware systems is still prohibitive for everyday use by an application domain expert who has no background in middleware development, Grid computing or even computer science. On the other hand, an engineering solution to a concrete problem is often a team effort undertaken by a number of involved engineers, and other non-IT personnel. Current support for collaborative software development is often limited to the use of CVS, email and conference calls. Such offer only limited support to ease the entry of engineers not trained in formal software development processes into the Grid.

As application the optimization of the Bridgman casting process of gas turbine blades was chosen. The highest gas turbine efficiency is achieved today with single-crystal (SX) or directionally solidified (DS) blading material, commonly casted in a Bridgman furnace (Figure 5). The Bridgman process is controlled by time dependent parameters (withdrawal speed, heater temperatures), which are ideal for the application of numerical optimization [6, 7, 8]. In addition, the blade casting is the most expensive process during the manufacturing of a turbine. This reduction of production cost by optimization attracts interest of commercial users in industry.

The paper is organized as follows: In the next section related work on Grid Computing and non-linear optimization is briefly discussed. Afterwards the Distributed Polytope algorithm is introduced used to solve a problem from metal casting. A Grid-based Problem Solving Environment designed for the solution of this kind of problems is presented in Section 4. A prototypical implementation of a Grid-based PSE is used to solve a problem from metal casting as described in Section 5. Preliminary results of this optimization problem followed by a summary and some areas for future work conclude the paper.

2. Related Work

Development of direct optimization algorithms for simulation-based optimization is under research for several decades. Simplex-based methods were successfully applied to engineering problems and demonstrated their robustness [15, 28]. Since the computation of objective and constraint function values of the Simplex- (or Complex-) points are independent the development towards a parallel strategy is obvious and resulted in approaches like the Parallel Direct Search (PDS, [24]) and Multidirectional Search (MDS, [25]). A subset of direct methods can be subsumed under the class of Pattern Search algorithms (see e.g. [14]) which are also easy to parallelize. The problem of handling infeasible solutions within any optimization strategy can be solved by using e.g. penalties (see e.g. [17]) or repair mechanisms as known from Genetic Algorithms, which is the preferred approach in this case [21]. Repairing infeasible solutions preserves implicit information on the search space and allows exploration of regions in the search space maybe not covered by the search strategy of the optimization method.

Supporting workflows – especially concerning the complex processes of pre- and postprocessing of simulation models and simulation-based optimization – with software systems and especially service-oriented architectures realized with web services have received considerable attention in both academia and industry. Several other research projects try to cope with similar subjects in related fields.

Supporting business processes with software systems and especially service-oriented architectures realized with web services have received considerable attention in both academia and industry. Several other research projects try to cope with similar subjects in related fields.

The Geodise project (see [23, 29]) focuses on optimization, design and fluid dynamics, especially in aerodynamics. Its main goal is to provide a distributed problem solving environment (PSE) for engineers working in the mentioned fields by utilizing e.g. MATLAB and adding Grid functionality to it. Although first Geodise implementations were based on the Globus Toolkit version 2, the core Geodise Toolbox is now part of the managed program of the *Open Middleware Infrastructure Institute* (OMII) [19].

A Grid-enabled problem solving environment for engineering design where distributed parties are able to collaborate has been introduced by Goodyer et al. [9]. The system makes use of the gViz Library [4] which allows collaborative visualization on the Grid and provides the user to start Grid jobs on Globus Toolkit based hosts. The main focus is put on collaborative application steering and result visualization of given simulation problems.

The P-GRADE Portal (see [22]) aims to be a workflow-oriented computational Grid portal, where multiple clients can collaboratively participate in design, development and execution of a workflow as well as multiple Grids may be incorporated in the workflow execution. The P-GRADE Portal is based on the Globus Toolkit version 2 for file transfer operations and job execution, the workflow execution is done by a proprietary implementation. P-GRADE neither uses Grid service and business process standards such as BPEL, nor does the proposed collaborative editing approach support real time collaboration on a process in an on-line meeting style.

The mentioned software systems are examples for the large variety of problem solving environments, collaborative Grid application systems and collaborative workflow development systems. However, none of the mentioned systems provides both a problem solving environment for engineering problems as well as sophisticated support for the collaborative software development process for Grid applications and their execution in a service-oriented Grid environment. Collaboration support often relies on out-of-band collaboration and synchronization techniques such as exchanging e-mail or CVS like server based communication.

3. The Distributed Polytope Algorithm

This following sections introduce the design of a distributed optimization algorithm which is based on traditional, direct optimization methods, their parallel extensions as well as aspects from the class of Genetic Algorithms. Beyond the principle design of the algorithm presented here, a detailed performance analysis regarding properties like parallel speedup and efficiency is presented in [1, 2].

The proposed distributed direct optimization algorithm is based on the concepts of both the simplex-based parallel direct search method [5] for bound-constrained and the sequential Complex Box method ([3, 10]) for constrained nonlinear optimization. Since constraint handling is of great importance, especially when taking into account the computational cost of each individual evaluation of objective and constraint functions, the algorithm tries to retrieve knowledge about the search space (feasible and infeasible regions) by integrating infeasible solutions into the search process. This is done by repairing infeasible solutions using a parallel algorithm for moving them into a feasible region of the search space. The aspect of repairing infeasible solutions is taken from Genetic Algorithms were this technique can be used to assure a feasible population after recombination/mutation operations.

In contrast to earlier applications of the Distributed Polytope method to simulation-based problems in engineering (see e.g. [11, 20]) this repair mechanism is also applied to solutions from the polytope which were 'out of bounds' after the reflection operations in exploration instead of setting the violated bound to the maximum/minimum allowed value. This way the approaches of using penalty functions and repairing infeasible solutions as constraint handling can be integrated in a problem formulation and solved by the Distributed Polytope algorithm.

Additionally, the problem of finding the global optimum is tackled by a hybrid approach combining the more global simplex/complex methods with (parallel) local search strategies to overcome the weakness of relatively slow local convergence of simplex/complex methods. The basic idea of the algorithm is the adaptation of the search strategy according to problem size and resources. To achieve this, different parameters of the algorithm are provided determining the degree of parallelism, i.e., the number of parallel constraint and objective function evaluations per iteration and the multiplicity of search directions.

Both previously mentioned methods are based on geometrical operations (reflection, contraction) performed on the vertices of an (at least) $(n+1)$–dimensional polyhedron in the n–dimensional solution space of the optimiza-

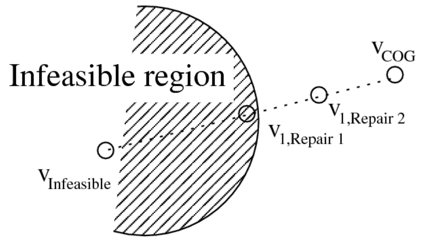


Figure 1. Repair of the infeasible solution $v_{Infeasible}$ using a parallel binary search along the direction towards the weighted center of gravity v_{COG}. The first attempt yields the infeasible solution $v_{1,Repair1}$, the second parallel attempt the feasible solution $v_{1,Repair2}$.

tion problem. The basic parameters are the size $s \geq n+1$ of the polytope (the number of vertices), the number of vertices e which are modified using reflection and contraction, and the 'look ahead' factor which controls the number of new vertices l generated by reflecting or contracting one vertex. Additionally, the point on which the vertices are reflected can also be varied. The vertices can be reflected on the best solution or the weighted center of gravity (vertices weighted with the value of the objective function of the vertex). In Figure 2, various alternatives are illustrated. Different settings of the aforementioned parameters yield different search strategies by introducing additional search directions. It can be seen that reflection on the best vertex restricts the search to the direction of this vertex while reflection on the weighted center of gravity allows searching in all directions given by the vertices of the polytope.

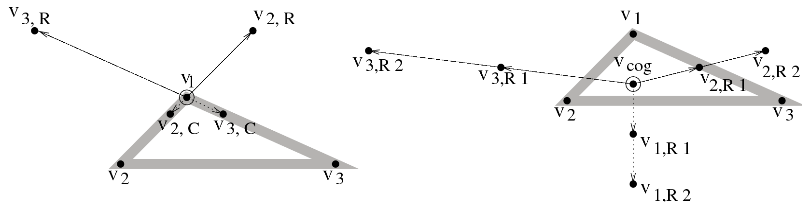


Figure 2. Reflected ($v_{i,R}$) and contracted ($v_{i,C}$) solutions when reflecting on a vertex (v_1, left) or on the center of gravity (v_{COG}, right).

The strategy used throughout the algorithm to repair infeasible solutions is depicted in Figure 1. The characteristics of the feasible region obviously determine the effort of repairing an infeasible vertex. In Figure 3, the scheme of a line search in a search space with discontiguous feasible regions is shown. The number of repair steps depends on the topology of the search space and the process of the line search. Since the topology of the search space of a general simulation-based problem is unpredictable, a precise number of infeasible solutions and repair steps is generally unavailable prior to the optimization.

The algorithm used for the solution of the subsequently presented optimization problems comprises the following steps (n denotes the dimension of the optimization problem, p the number of available workstations in the network):

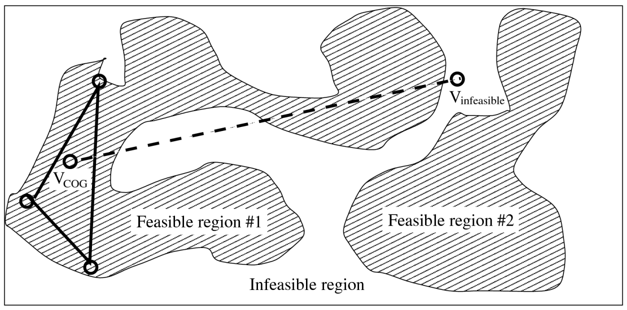


Figure 3. Scheme of a repair using a line search on a search space with unconnected infeasible regions.

1 **Initialization:** The starting polytope consisting of $s > n + 1$ randomly generated solutions is built and the constraints are evaluated on p workstations in parallel. Infeasible solutions are repaired using a parallel binary search directed towards the weighted center of gravity of feasible solutions.

2 **Exploration:** The $e \leq s$ worst solutions are reflected on, respectively moved towards the weighted center of gravity (reflection/contraction). Each of these reflections is performed l times in parallel (see Figure 2) yielding $2{\cdot}e{\cdot}l$ new solutions. All solutions are evaluated on p workstations in parallel. Infeasible solutions are repaired using parallel binary search. The s solutions for the polytope of the next iteration are selected from these solutions.

3 **Local Search:** When the exploration is terminated (e.g., after the maximum number of iterations) a parallel local search starts from the best solution. It evaluates in parallel p random solutions in an environment with radius r around the best solution. The radius is reduced if the local search fails to find a better solution. Infeasible solutions are rejected instead of being repaired as in the previous phases. The local search stops after a given number of iterations or when the improvement is less than a given ϵ.

This approach has been already successfully applied to several problems from engineering domains other than metal casting: in groundwater management [11] and in the aircraft industry [20]. Analyzes concerning speedup and efficiency have been performed to evaluate scalability [1].

This schematic overview of the Distributed Polytope algorithm exhibits some principle similarities between simplex-, complex- and polytope-based methods and Genetic Algorithms: Algorithms of both classes are based on a set ('population' vs. 'simplex/complex/polytope') of solutions ('individuals' vs. 'vertices'). In a single iteration a subset of elements is selected and some new elements are calculated with special operators ('recombination/mutation' vs. 'reflection/contraction'). All the elements of this set can be evaluated independently from each other and therefore they are particularly suitable for distributed computation. Nevertheless it must be remarked that large population (or polytope) sizes—as typically used in Genetic Algorithms—should be carefully used in simulation-based optimization since any of the objective and constraint function evaluations may take several hours. The number of simulations together with the available degree of parallelism determines the overall runtime of the optimization. Since the (heuristic) determination of a search direction by reflection/contraction as utilized in the Distributed Polytope algorithm seems to be a good compromise between robustness and the necessary number of simulations it is selected for further analysis and application to problems in engineering. Simplex-based as well as Genetic Algorithms can be applied in distributed environments such that they can utilize a larger number of computational resources in order to speed up the overall optimization time (see e.g. [7]). Hence, a Grid-based PSE like the one presented in the next section is useful for algorithms from both classes.

4. A Grid-Based Environment for Simulation-Based Optimization of Casting Problems

In this section, a simplified view on a sample application from an engineering domain is presented to motivate the need for support in the distributed software development process of a Grid software system for engineering applications. The concrete use case comes from casting, a sub-domain of metal forming. Only those parts relevant to the Grid are briefly sketched; they do not reflect the entire complex field of metal forming. For more information regarding the complexity involved in collaborative engineering particularly in the field of metal forming and casting, the reader is referred to e.g. [18, 27].

In the metal casting industry, customers' quality requirements, e.g. allowed tolerances in a casting product's geometry compared to the specification, are constantly increasing. Therefore, the use of numerical simulation and simulation-based optimization is gaining importance, since the creation of prototypes is in many cases too expensive and prohibitively time consuming. The benefit of this 'virtual prototyping' based on numerical simulation is constrained by the accuracy (i.e., the difference between simulated and the real physical behavior of a casting process) of the simulation environment. Both the creation and use of the simulation as well as optimization application require great expertise in the metal casting domain. Furthermore, applying numerical simulation for this purpose introduces an extremely high demand for computational capacity since a single—sufficiently precise—simulation run typically lasts several hours up to days. Since many small and medium sized engineering enterprises are not capable of acquiring and maintaining high performance computing resources, outsourcing of computationally demanding tasks is necessary. Grid computing promises to offer the infrastructural components to realize this outsourcing activity as easy as plugging into the electrical power Grid. However, currently the implementation of a Grid application still requires these firms to involve Grid specialists to adapt and maintain their applications in a Grid environment.

To summarize, the utilization of numerical simulation in the casting industry demands a variety of competencies:

- knowledge about the physical properties of casting in industrial practice (casting engineer)
- modeling a casting engineering process for simulation (casting engineers together with IT specialists)
- adapting existing simulation software to the Grid (Grid specialists consulting the casting engineers)
- setting up and maintaining a simulation and/or optimization environment for the engineers' customers (Grid specialists, casting engineers and their customers)
- interpreting a simulation's result (casting engineer and customer).

These requirements lead to a software platform which enables the integration of the aforementioned competencies and resources during the software design process. Since most of the possible users of simulation in the casting industry are small to medium enterprises (SME), lacking at least one of the requirements, the Grid software platform must be able to facilitate both renting computational resources on demand as well as the collaborative involvement of Grid experts, casting engineers and their customers.

As a concrete sample scenario, we introduce the engineering process of collaborative development of a metal casting model. The following two services form the most important building blocks for the overall problem solving process to be deployed on a Grid system.

The Distributed Polytope Service. This service is an implementation of the distributed polytope optimization algorithm as introduced in 3. During its runtime, it requires an a priori unknown number of evaluations of both an objective function and corresponding constraint functions, in this case calculated by the metal casting simulation software CASTS [13]. The service has to save its state each time an evaluation request occurs, and it passes the data set which is to be evaluated to the process execution engine instead of directly invoking the simulation service. Considering these conditions, the service was implemented by utilizing the web service resource framework (WSRF), which allows the creation of stateful web service resources. Beside a service operation which allows a client to set necessary parameters needed by the polytope algorithm, the only Grid service operation `iterate(IterateRequest)` takes care of starting and restarting the algorithm at the appropriate position – according to its internal state and according to the input data inside the `IterateRequest` data structure. A resulting data set is returned immediately after invoking the operation, telling the process execution engine if further evaluations are needed or if the polytop algorithm reached a predefined stop condition.

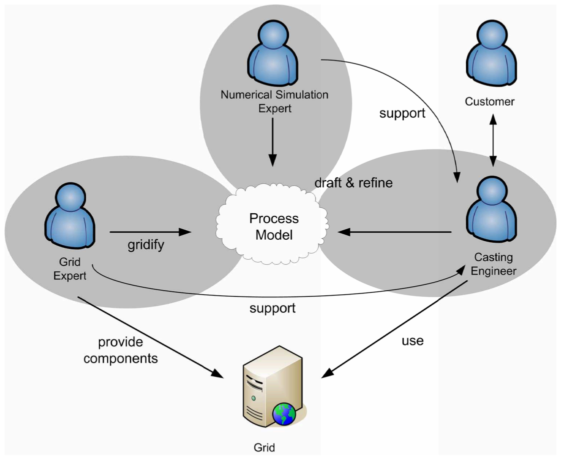


Figure 4. Persons, competencies and their functions in the collaborative process of preparing the software environment, simulating and optimizing a process model.

The CASTS Service. The main purpose of this service is to wrap the metal casting legacy software CASTS as a Grid service. However, the *Casts Service* does not only provide a service-wrapped version of CASTS, but it also takes care of the following operations: It is capable of modifying the input model of the casting process according to a set of parameters passed to the service. This parameter set is the input received from the distributed polytop algorithm. The service executes the CASTS legacy application on a number of different execution platforms. In this case, a 128-node cluster computer with two 64-bit AMD Opteron CPUs and 2 GB main memory per node was utilized, leading the execution subsystem to incorporate the local resource manager Torque [26] and the scheduling system Maui [16]. The execution state of a cluster job is monitored and exposed by the Casts Service. The execution subsystem is highly modularized so that the service also works on single workstations without local queuing/scheduling. The service also provides functionality to evaluate the simulation result (which is done by CritCASTS, a legacy software system bundled with CASTS) and determining the objective function value as well as the constraint function values.

An overall view of the collaborative and distributed development scenario is shown in Figure 4. The gray zones mark the network domains of the different experts, they are geographically distributed, and their collaboration takes place via the shared and synchronized process model.

5. Case Study

5.1 Bridgman Casting Process: Model and Simulator

Turbine blades of modern aircraft and power plants are made of Ni-base superalloy and are commonly produced by directional solidification (DS) or single crystals (SX) in a Bridgman furnace (see Figure 5). A directional heat flow is created by withdrawing the shell mould of the turbine blade out of the heating zone into a cooling zone. The strong temperature gradient at the interface between heating and cooling zone leads to a directional solidification. Beside the simplicity of the Bridgman principle the optimization of all process parameters is complex for real blade geometries [8]. Technically relevant casting parameters, such as heater's temperature and withdrawal velocity, are currently determined by series of expensive experiments.

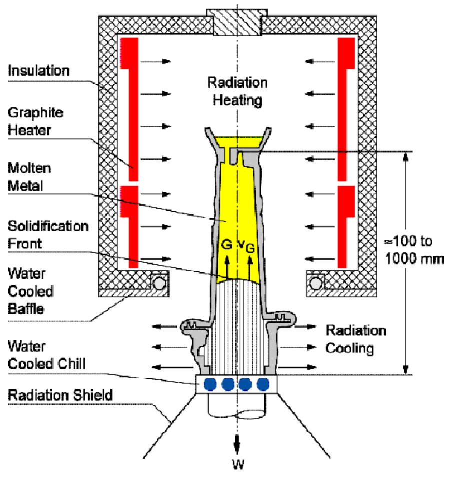


Figure 5. Schematic description of a Bridgman furnace used for directional solidification

The hybrid FE/CV programm CASTS (Computer Aided Solidification TechnologieS) [13], is used to predict numerically the transient temperature response during the Bridgman casting process. CASTS calculates transient temperature distributions in mold, core and alloy, taking into account both latent heat release as a function of fraction solid, and heat transfer resistance at material interfaces. The main output is the temperature and heat flux field. Based on this data, temperature gradients and defect maps can be calculated for each set of input process parameters, which are the basis for the evaluation of the turbine blade.

5.2 Optimization Variables and Target Function

Goal of the optimization was an improved withdrawal profile for the Bridgman process of a cluster of simplified turbine blades. The whole withdrawal process is parameterized by eleven bounded design variables representing the withdrawal velocities at different times.

The simulation results were evaluated applying four criteria given as follows:

- the probability of surface defects formation, so called freckles. The freckle probability was estimated based on the temperature gradients calculated by the FEM-simulator;
- the degree of curvature of the solidification front. The solidification front should be as horizontal as possible in order to achieve a high quality directional solidification.
- the ratio G/v (temperature gradient over solidification speed) must be greater than a critical value. At this critical value the transition from the desired columnar dendritic growth to an undesired equiaxed grain structure takes place.
- the process time.

In order to achieve a better combination of the four criteria, a new formulation has been developed for the first three optimization criteria, the freckle probability, the curvature of the solidification front and the G/v ratio. These criteria are evaluated by counting the number of 'bad' nodes, i.e., nodes of the finite element mesh with freckle probability above zero, a curvature of the solidification front above $20\,^{\circ}$or a G/v ratio below $600\,\mathrm{Ks/cm^2}$. The criteria can be tuned by changing the limits $(0{,}20\,^{\circ},\ 600\,\mathrm{Ks/cm^2})$. A great advantage of this new criteria formulation is that these three criteria can now be easily combined due to there similar definition by the number of bad nodes.

As an objective function the overall process time has to be minimized. The constraints are integrated into the objective function by using a two-stage objective function: As long as one constraint is not fulfilled the objective function is the sum of the number of bad nodes and the process time in seconds. If the latter is below 5,000 s a constant value of 5,000 is used to focus the optimization to the fulfillment of the other constraints. If all constraints are fulfilled, the simulation time in seconds becomes the objective function and the optimization searches for a further reduction of the process time.

This handling of constraints is similar to the use of penalty functions. In contrast to the usual approach (adding a penalty term to the objective function and solving an unconstrained problem) in this context the objective function integrates a kind of penalty terms and infeasible solutions (i.e., solutions violating bounds on the design variables) are although repaired.

5.3 Results

As a first step towards a complete Grid-based environment for simulation-based optimization of casting processes, the Distributed Polytope algorithm was applied in a parallel testbed (up to 300 CPUs available in a cluster system) to the 11-dimensional test problem. The essential parameters were set to $e = 10, l = 3$ yielding $2el = 60$ newly computed solutions per iteration, using a polytope size $s = 2n = 22$. e and l were set to the respective values to assure a simultaneous search in many directions ($e = 10$ directions per exploratory step) but with limited ($l = 3$) 'look ahead' and hence a limited extent of the polytope.

In Figure 6 the results of the Distributed Polytope algorithm running on 20 CPUs are compared to those from Metamodel Assisted Derandomized Evolution Strategy (MADES) utilizing 4 CPUs. The MADES was optimized over several years for the optimization of such engineering processes. A detailed description can be found in [6] and latest results were published in [12]. It can be seen that the results (7,447 from the Distributed Polytope vs. 7,513 from MADES) as well as the trajectories of the objective function values during the optimization runs are comparable. Both algorithms are not able to improve beyond a certain quality of the objective function which is reached by MADES substantially earlier ($t = 15,000\,\mathrm{s}$) than by the Distributed Polytope ($t = 21,000\,\mathrm{s}$). It must be analyzed whether the switching between exploration and termination phase can be adjusted in order to avoid this behavior. It must be noted that there was no tuning concerning the parameter set of the Distributed Polytope and the shown results were computed in a single run. In order to validate the solution quality additional runs must be performed. Furthermore, parameter studies for the main parameters e and l as well as the degree of parallelism must be performed to evaluate the behavior of the algorithm in detail.

6. Conclusions

In this paper, a grid infrastructure is introduced, which simplifies the use of distributed, parallel numerical optimization by bioinspired and related optimization strategies. As showcase the optimization of an industrial casting process was chosen. Based on the Grid Computing toolkit GLOBUS, Grid Services for the Distributed Polytope optimization algorithm and the casting

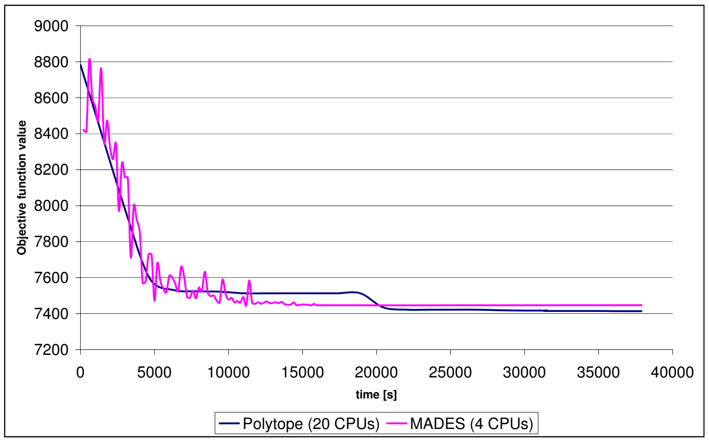


Figure 6. Comparison of the Distributed Polytope algorithm (using 20 CPUs) with the MADES algorithm (running on 4 CPUs) when solving a problem based on the CASTS simulation system.

simulation package CASTS were developed and tested. This Grid Service-based environment was applied to solve a test problem from metal casting.

A Distributed Polytope Algorithm was used as optimization strategy. The algorithm integrates aspects from traditional deterministic sequential simplex-based methods, parallel search strategies and non-deterministic bioinspired methods like Genetic Algorithms. This combination of approaches was used for the design of the algorithm in order to satisfy requirements concerning mathematical properties and runtime behavior specific for solution processes in the simulation-based optimization of problems from engineering, in this case from metal casting.

As first application the optimization of process parameter of casting a geometrically simplified gas turbine blade in a Bridgman process was selected. The results were compared to the distributed evolutionary strategy MADES and demonstrate the applicability of a scalable distributed optimization algorithm integrated into a distributed, Grid-based infrastructure for numerical optimization of industrial processes.

As a next step different degrees of parallelism as well as different settings for other parameters of the Distributed Polytope algorithm will be evaluated concerning quality of result and runtime of the optimization. This will be the basis for a detailed comparison to other optimization strategies. From the software engineering point of view the integration of the complete workflow from modeling, calibration of the model to optimization is planned to be integrated within the Grid-based environment.

Acknowledgements

Parts of the work presented in this paper are supported by a grant from the German Ministry of Education and Research (BMBF) (D-Grid initiative, In-Grid project).

References

[1] T. Barth, B. Freisleben, M. Grauer, and F. Thilo. Distributed solution of optimal hybrid control problems on networks of workstations. In *Proc. IEEE International Conference on Cluster Computing (CLUSTER'2000)*, pages 162–169, Chemnitz, Germany, 2000.

[2] T. Barth, B. Freisleben, M. Grauer, and F. Thilo. A scalable algorithm for the solution of simulation-based optimization problems. In *Proc. International Conference on Parallel and Distributed Programming Techniques and Applications (PDPTA'2000)*, pages 469–475, Las Vegas, Nevada, USA, 2000.

[3] M. Box. A new Method of Constrained Optimization and a Comparison with other Methods. *Computer J.*, 8:42–52, 1965.

[4] K. Brodlie, D. Duce, J. Gallop, M. Sagar, J. Walton, and J. Wood. Visualization in Grid Computing Environments. In *Proc. IEEE Visualization*, pages 155–162, Austin, Texas, USA, 2004.

[5] J. Dennis and V. Torczon. Direct search methods on parallel machines. *SIAM J. Optim.*, 1:448.474, 1991.

[6] M. Emmerich and J. Jakumeit. Metamodel-assisted optimisation with constraints: A case study in material process design. In *EUROGEN 2003*, Barcelona, Spain, 2003.

[7] M. Emmerich, M. Schallmo, and T. Bäck. Industrial applications of evolutionary algorithms: A comparison to traditional methods. In I. Parmee et al.: *Optimisation in Industry*, Springer, Berlin, pages 304–314, 2001.

[8] M.S.G. Laschet and N. Hofmann. Optimization tools for casting processes. In *Proc. 7th Conference on Casting, Welding and advanced Solidification*, pages 1095–1102, San Diego, USA, 1998.

[9] C. Goodyer, M. Berzins, P. Jimack, and L. Scales. A Grid-enables Problem Solving Environment for Parallel Computational Engineering Design. *Adv. Eng. Software*, 37(7):439–449, 2006.

[10] M. Grauer. *Verfahrenstechnische Berechnungsmethoden*. Chapter Optimierung verfahrenstechnischer Systeme, pages 127–129, Verlag Chemie, 1987.

[11] M. Grauer, T. Barth, S. Kaden, and I. Michels. Decision support and distributed computing in groundwater management. In *Water Industry Systems: Modelling and Optimization Applications*, pages 23–38, 1999.

[12] J. Jakumeit, M. Emmerich, and F. Hediger. Inverse modeling and numerical optimization of heater temperatures in a bridgman process. In *Proc. of Modeling of Casting, Welding and Advanced Solidification Processes XI*, pages 1019–1026, 2006.

[13] G. Laschet, J. Neises, and I. Steinbach. Micro- Macrosimulation of casting processes. 4^{ieme} *école d'été de Modélisation numérique en thermique*, page 1.42, 1998.

[14] R. Lewis and V. Torczon. Pattern search algorithms for bound constrained minimization. *SIAM J. Optim.*, 9(4):1082–1099, 1999.

[15] R. Lewis, V. Torczon, and M. Trosset. Direct search methods: Then and now. *J. Comput. App. Math.*, 124(1-2):191–207, 2000.

[16] Maui cluster scheduler. `http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php`.

[17] Z. Michalewicz and D. B. Fogel. *How to solve it: modern heuristics*. 2nd edition, Springer, 2004.

[18] T. Nguyen and V. Selmin. Collaborative Multidisciplinary Design in Virtual Environments. In *Proc. 10th International Conference on CSCW in Design*, pages 420–425, Nanjing, China, 2006.

[19] Open Middleware Infrastructure Institute (OMII). `http://www.omii.ac.uk/`.

[20] G. Schneider, F. van Dalen, T. Barth, H. Hörnlein, and M. Stettner. Determining wing aspect ratio of a regional aircraft with multidisciplinary optimisation. In *Proc. CEAS Conference on Multidisciplinary Aircraft Design and Optimization*, Cologne, Germany, 2001.

[21] M. Schoenauer and S. Xanthakis. Constrained ga optimization. In *Proc. 5th International Conference on Genetic Algorithms*, pages 573–580, Urbana-Champaign, IL, USA, 1993.

[22] G. Sipos and P. Kacsuk. Collaborative Workflow editing in the P-GRADE. In *Proc. International Scientific Conference microCAD 2005*, Miskolc, Hungary, 2005.

[23] W. Song, Y.-S. Ong, H.-K. Ng, A. Keane, S. Cox, and B. Lee. A service-oriented approach for aerodynamic shape optimization across institutional boundaries. In *Proc. 8th International Conference on Control, Automation, Robotics and Vision*, Kunming, China, 2004.

[24] V. Torczon and M. Trosset. From evolutionary operation to parallel direct search: pattern search algorithms for numerical optimization, 1998.

[25] V. J. Torczon. ulti-Directional Search: A Direct Search Algorithm for Parallel Machines. PhD thesis, Housten, TX, 1989.

[26] Torque resource manager. `http://www.clusterresources.com/pages/products/torque-resource-manager.php`.

[27] S. Woyak, H. Kim, J. Mullins, and J. Sobieszczanski-Sobieski. A Web Centric Architecture for Deploying Multi-Disciplinary Engineering Design Processes. In *Proc. 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Albany, New York, USA, 2004.

[28] M. Wright. Direct search methods: Once scorned, now respectable, 1995.

[29] G. Xue, W. Song, S. Cox, and A. Keane. Numerical Optimization as Grid Services for Engineering Design. *J. Grid Comput.*, 2(3):223–238, 2004.

MAINTAINING SOLUTION DIVERSITY IN A HYBRID EVOLUTIONARY ALGORITHM FOR EPR-BASED SPIN LABEL CHARACTERIZATION OF BIOSYSTEM COMPLEXITY

Aleh A. Kavalenka, Janez Štrancar
Solid State Physics Department
Jožef Stefan Institute, Ljubljana, Slovenia
{oleg.kavalenka,janez.strancar}@ijs.si

Abstract The paper presents new ideas of maintaining population diversity in a hybrid evolutionary algorithm used for spectrum parameter optimization when characterizing the complexity of biological systems. Recent modification of the evolutionary algorithm (EA) by introducing the 'shaking' operator enabled to maintain solution diversity and speed-up calculations by factor of 5–7.

Keywords: Diversity, Hybrid evolutionary algorithm, Optimization, Shaking

1. Introduction

Preserving the genetic diversity throughout evolutionary algorithm generations is a key point to make the algorithm capable of revealing multiple solutions in a complex multi-dimensional search space [10].

We apply an evolutionary optimization algorithm to study the complexity, one of the basic properties of natural biological systems [6, 15]. Qualitatively the complexity can be described by the number of (biochemical or biophysical) patterns/solutions that coexist in a system. A pure system can be characterized only by one solution, whereas in complex systems several distributions of solutions may exist.

Electron paramagnetic resonance (EPR) spectroscopy in combination with nitroxide spin labeling (SL) has proven to be a powerful technique for the exploration of heterogeneity and motion in biological systems [3]. However, to determine the picture of the actual complexity of the biological system, a special methodology that includes advanced spectrum analysis and inverse-problem solving techniques has to be applied [16]. Such an analysis is based on mathematical modeling, spectrum fitting, and spectral parameter optimization.

To present multiple results, a special method of solutions condensation called GHOST was developed [15]. GHOST incorporates solution density filtering, χ^2 goodness filtering, solution-space slicing, and domains determination.

This advanced approach named Hybrid Evolutionary Optimization (HEO) was shown to be powerful enough to study complex heterogeneous systems although the computational demand appeared to be an obstacle for wider usage of the method. To obtain a reliable result, the HEO procedure has to be executed 200 times. Each particular run implies 100 generations with population size of 300 candidate solutions. Since an average operator performs up to 10 spectrum calculations, HEO on average spends 60 million spectrum calculations. As a single spectrum calculation takes around 10 ms on a 1 GFLOPS processor, this results in 200 hours of computer time spent for a single characterization. Our aim was to improve the solution diversity of a single HEO run by maintaining genetic diversity throughout the HEO routine.

2. Theory and Methodology

2.1 EPR Spectrum Analysis

The scheme of the spectrum analysis and inverse-problem solving is presented in Figure 1. Since EPR spectrum modeling has been already discussed [16], we only present spectral parameters that are involved in calculations. Taking into account the superposition of motional/polarity patterns, the set of parameters $\vartheta, \varphi, \tau_c, W, p_A, Prot$ is expanded for the number of spectral components N_c. In addition, there are $N_c - 1$ weights d of these spectral components. Altogether, there are $7N_c - 1$ spectral parameters, which have to be resolved by the optimization routine. The resolution limit of SL-EPR assumes up to 30 parameters and this allows at most 4 spectral components.

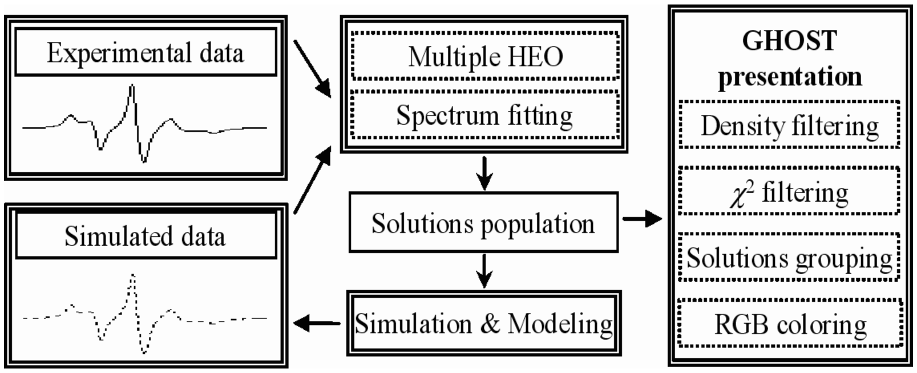


Figure 1. EPR spectrum analysis scheme.

The goodness of fit (objective function) is the reduced χ^2 criterion:

$$\chi^2 = \frac{1}{N-p} \sum_{i=1}^{N} \frac{(y_i^{exp} - y_i^{sim})^2}{\sigma^2}, \tag{1}$$

where y^{exp} and y^{sim} are the experimental and simulated data, respectively, σ is the standard deviation of the experimental points, N is the number of spectral points, and p is the number of model parameters.

2.2 Hybrid Evolutionary Optimization

HEO is a combination of the Genetic Algorithm (GA) and the Local Search Downhill-Simplex algorithm. The optimization routine starts with a random initialization of solutions population and continues with the tournament selection and application of genetic operators for 100 generations. The 3-point crossover with probability of 0.7 and uniform mutation with probability of 0.01 are applied together with certain knowledge-based operators and local improvements (performed with Downhill-Simplex [5] with probability of 0.002). The elite set (2 % of the population size) is used to preserve the best found individuals. One HEO run assumes 100 generations of GA. GA population size is 300 individuals. In 200 HEO runs a group of 200 best parameter sets (best from each run) is accumulated and then filtered, grouped, and graphically presented with the GHOST condensation algorithm.

Parameter Search Space. The optimization process searches for the minima in the landscape of the parameter search space, which may contain both local and global minima. Our particular requirement is that the optimization routine should be able to find global minimum(a), which can be of different types, i.e., well-defined minima of type B or a flat valley minima of type A (see Figure 2). The convergence to the minima of type B (discrete problems) has to be provided as well as population diversity has to be maintained to enable the optimization procedure to fully reveal the minimum valleys (in case of continuous problems) already in a single run.

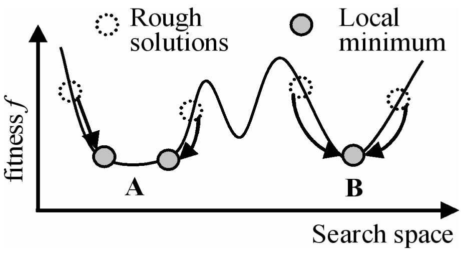


Figure 2. Schematic presentation of one dimension of the parameters search space and the effect of the local mutation procedure responsible for fine-tuning.

Maintaining the Population Diversity. Simple genetic algorithm (SGA) [7] is suitable for finding the optimum of a unimodal function in a bounded search space. However, both experiments and analysis show that the SGA cannot find the multiple global maxima of a multimodal function [7, 12, 13] or a function with a flat global minimum, which is an extreme limit of the multimodal function. This limitation can be overcome by mechanisms that create and maintain several subpopulations within the search space, referred to as 'niching methods': sequential niching methods [1]; parallel niching methods (sharing [8], crowding [12], clearing [13] etc.); speciation methods [11, 14], clustering [17]; multi-population methods [2]). Another way to find multiple optima is to make several runs of an ordinary GA. In each run the GA converges to different optima. Thus, several optima are found [4]. This strategy was initially implemented in HEO-based approach. Since the methods that assume creating subpopulations do not match with our specific problem, we chose the sharing parallel niching method for maintaining diversity within a single run.

Sharing. Sharing [8, 12] requires that fitness is shared as a single resource among similar individuals in the solutions population. The fitness sharing method modifies the search landscape by changing the fitness function, i.e., the value of χ^2, in densely-populated regions. As a result, the population becomes better distributed in the search space. The fitness function f is modified as follows:

$$f'(j) = \frac{f(j)}{\sum_{i=1}^{n} sh(d[i,j])}, \tag{2}$$

where the sharing function sh is a function of distance $d[i,j]$ between two population elements. It returns '1' if the elements are identical and '0' if they cross some threshold of dissimilarity, specified by constant σ_{share}:

$$sh(x) = \begin{cases} 1 - (\frac{x}{\sigma_{share}})^{\alpha} & \text{if } x < \sigma_{share}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Here α is a constant, which regulates the shape of the sharing function. Fitness sharing is demonstrated in Figure 3(a).

Shaking. The second proposed approach for improving the solution diversity is 'shaking' operator. The shaking operator provides small Gaussian-like deviations to the spectral parameters before the crossover operator is applied (see Figure 3(b)). The error bars indicate the width of Gaussian probability distribution of these deviations. The standard relative uncertainties of the spectral parameters $\{\vartheta, \varphi, \tau_c, W, p_A, prot, d\}$ are $\{0.02, 0.02, 0.04, 0.035, 0.035, 0.04, 0.02\}$, respectively, which follow average uncertainties that are found empirically for these parameters within the simulation model.

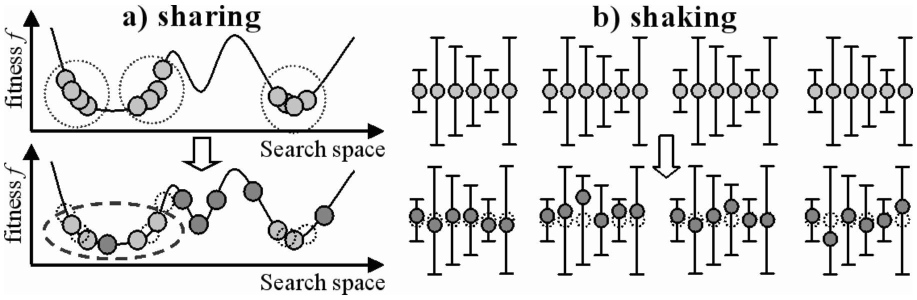


Figure 3. Schematic presentation of population diversity improving approaches: a) fitness sharing function; b) Gaussian shaking operator.

2.3 Projection Principle and GHOST Condensation

Large amount of solutions that come out from the multiple HEO runs is condensed and grouped together to construct a discrete or quasi-continuous description of the system. After solution filtering according to the local solution density and goodness of fit, the GHOST condensed results are presented in 2D cross-sections $\{S\text{-}\tau_c, S\text{-}W, S\text{-}p_A\}$ (see Figure 4). The color of any point (solution) in GHOST diagram is defined by RGB specification (where the intensity of each color component (red, green, blue) represents the relative value of the spectral parameters τ_c, W, p_A in their definition intervals $\{0\text{–}3\,\text{ns}\}$, $\{0\text{–}4\,\text{G}\}$, and $\{0.8\text{–}1.2\}$, respectively). This technique enhances the possibility to distinguish groups of solutions, and to explore optimized values of model parameters.

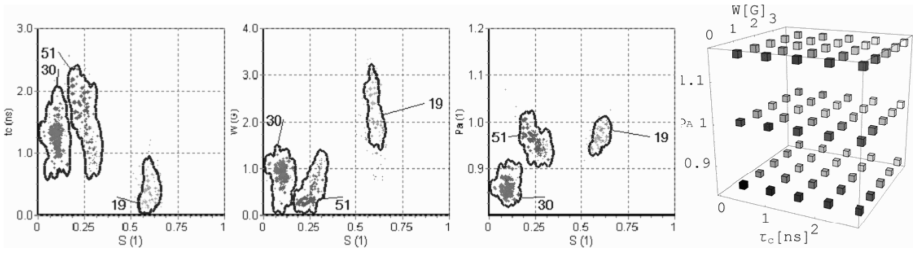


Figure 4. An example of the GHOST presentation (Spectra of spin labeled horse neutrophils membranes were fitted with EPRSIM BBW software and characterized using GHOST condensation procedure. RGB (red, green, and blue) color of any solution point codes the relative values of parameters τ_c, W, and p_A in their definition intervals.

The most important property of the GHOST algorithm is that there is no need to define the complexity (the number of different motional patterns) in advance – it comes as the result from the GHOST condensation and graphical presentation.

2.4 Evaluation Criteria

To judge the success of the modification of the HEO algorithm, the following criteria were selected: GHOST diagram quality (domains determination, parameters distribution); minimal fitness achieved χ^2_{min}, and fitness deviation $\sigma(\chi^2)$, that is 40 % of all selected χ^2 values; HEO runs contribution histograms; and maximal detected solution density ρ_{max}. To check the universality of the new algorithm we analyzed several types of EPR spectra: experimental (from membranes and membrane proteins) and synthetic (discrete and continuous).

3. Results and Discussion

Multiple Runs. The poverty of the final solution diversity becomes obvious if we reduce the number of HEO runs from 200 to 20 increasing the contribution of single run from 1 to 10 on average. The results for a typical experimental spectrum are shown in Figure 5, where the GHOST diagram (Figure 5(b)) and runs contribution histogram (Figure 5(c)) are compared with the initial GHOST diagram based on 200 runs (Figure 5(a)). The GHOST diagram based on 20 runs (Figure 5(b)) incorrectly describes the experimental system. Only a few HEO runs (first, seventh, ninth and seventeenth) contribute to the GHOST presentation (Figure 5(c)), whereas some other runs (third, fourth, tenth, etc.) do not contribute at all. Higher χ^2_{min} value (see Table 1) and uneven HEO runs contribution histogram corresponds to low solution diversity and to incorrect solutions domains determination (Figure 5(b)). Higher solution density (see ρ_{max} in Table 1) indicates solution crowding in the parameter search space.

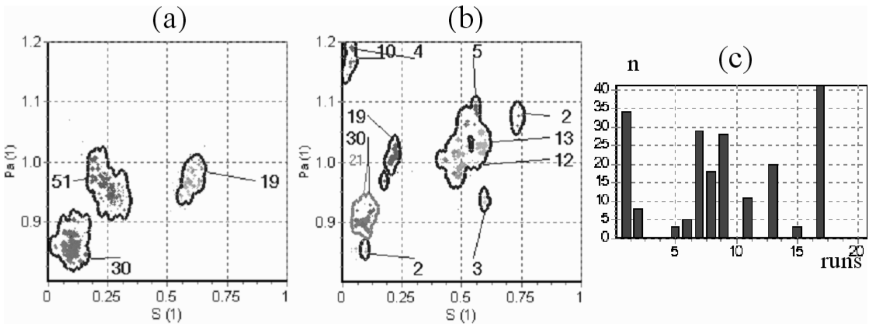


Figure 5. Characterization of spin labeled biological membrane: S-p_a GHOST cross-section: a) 200 HEO runs, one best solution is taken from a single run; b) 20 HEO runs, on average 10 solutions are taken from each run; c) HEO runs contribution histogram for the case of 20 runs (number of runs is shown along the x-axis and runs contribution number n along the y-axis).

Unsatisfactory result was also achieved for a synthetic 15-component spectrum that simulates a quasi-continuous distribution of spectral parameters (see Table 2 and Figure 6(b)). Poorly populated GHOST arises from the very uneven

Table 1. Optimization parameters after 200 and 20 runs for the membrane spectrum.

CRITERIA	200 RUNS	20 RUNS
χ^2_{min}	3.4	4.09
$\sigma(\chi^2)$	2.04	1.87
ρ_{max}	64.2	71.5

HEO runs contribution caused by solution crowding.

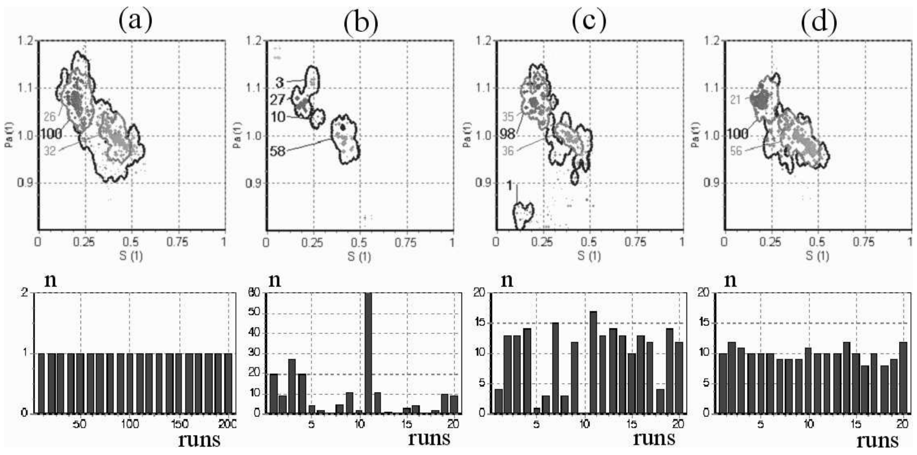


Figure 6. Comparing the results of different multi-run HEO by GHOST diagrams and HEO runs contribution histograms: a) 200 runs of original HEO; b) 20 runs of the original HEO; c) 20 runs of the modified HEO with the fitness sharing; d) 20 runs of the modified HEO with the shaking operator.

Table 2. Optimization parameters after running various variants of multi-run HEO (see also caption to the Figure 6).

CRITERIA	200 RUNS	20 RUNS	20 SHARING	20 SHAKING
χ^2_{min}	1.17	1.22	1.65	1.24
$\sigma(\chi^2)$	0.9	0.4	1.29	0.9
ρ_{max}	69.5	75.7	69	66.1

Sharing. The sharing approach was tested on a 15-component spectrum. The corresponding GHOST diagram better resembles the initial GHOST (compare Figures 6(c) and 6(a)). HEO runs contribution histogram (Figure 6(c)) is more even in comparison with the previous results. However, the distribution of χ^2

is worse than the initial (compare χ^2_{min} and distribution width $\sigma(\chi^2)$ in '200' and '20 sharing' columns of Table 2). Additional testing [9] showed that 20 HEO runs with fitness sharing are not enough to achieve the initial quality of systems characterization.

Grid Problem and Shaking. The cause of the solution crowding problem was found as the shortcoming of the three-point crossover GA operator. 'Genetic material', related to the promising parameters, copies and spreads in the population among individuals. After 20–30 generations, the population forms a 'grid' in the search space (see Figure 7) causing the loss of population diversity.

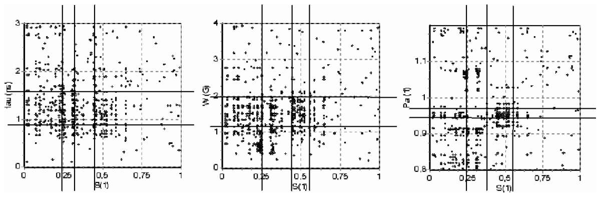


Figure 7. Demonstration of the 'grid' problem for three cross-sections of the search space.

Implementation of the shaking operator enabled the HEO algorithm to overcome the solutions crowding and to increase the population diversity in a single run (Figure 8).

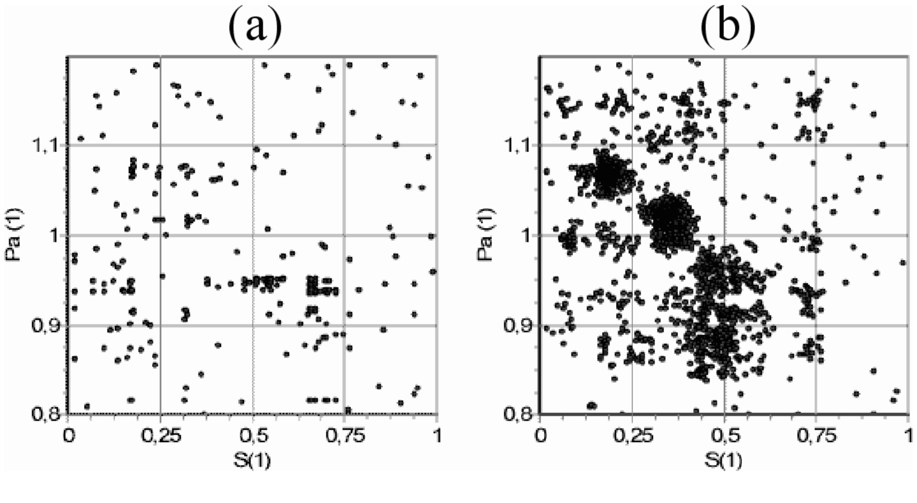


Figure 8. Single HEO run GHOSTs (population size 600): a) initial version of the algorithm with crowding problem – several solutions are crowded in different regions; b) version with shaking that maintains diversity – those solutions that were crowded before now spread over the flat minima region.

Modified with the shaking operator, HEO needs only 20 runs instead of 200, achieving the same quality of the systems characterization. This can be proven by comparing the quality of the GHOST diagrams (Figures 6(a) and 6(d)), by HEO runs contribution histogram (Figure 6(d)), and by good distribution of χ^2 (Table 2).

New algorithm enhanced with the shaking operator was further tested on several experimental and synthetic spectra in order to cover a wide range of possible systems related to discrete and continuous problems. The results of the characterization of four different examples [9] proved the capability of the modified HEO algorithm of resolving wide range of EPR spectroscopic data.

4. Conclusion

Maintaining solutions population diversity in EA by introducing a novel shaking operator reduced the computational demand of the original multiple HEO approach. Extensive testing of the modified multi-run HEO on various spectra that represent a wide range of possible applications proved its high efficiency. New modification of the optimization algorithm succeeded to keep high quality of system characterization, thereby considerably reducing the computational time by a factor of 5–7. With this successful modification, the application of advanced EPR spectrum analysis to complex biosystems, such as biological membranes and membrane proteins, became much more feasible.

Acknowledgment

We thank Dr. Bogdan Filipič for valuable discussions and advices about the extensions of the hybrid evolutionary algorithm.

References

[1] D. Beasley, D.R. Bull, and R.R. Martin. A Sequential Niche Technique for Multimodal Function Optimization. *Evol. Comput.*, 1(2):101–125, 1993.

[2] M. Bessaou, A. Pétrowski, and P. Siarry. Island model cooperating with speciation for multimodal optimization. In *Proc. 6th International Conference on Parallel Problem Solving From Nature*, pages 437–446, Paris, France, 2000.

[3] L. Columbus and W.L. Hubbell. A new spin on protein dynamics. *Trends Biochem. Sci.*, 27(6):288–295, 2002.

[4] P.J. Darwen and X. Yao. Every Niching Method has its Niche: Fitness Sharing and Implicit Sharing Compared. *Lect. Notes Comput. Sc.*, 1141:398–407, 1996.

[5] B. Filipič and J. Štrancar. Tuning EPR spectral parameters with a genetic algorithm. *Applied Soft Computing*, 1(1):83–90, 2001.

[6] B. Filipič and J. Štrancar. Evolutionary Computational Support for the Characterization of Biological Systems. In G.B. Fogel and D.W. Corne, editora, *Evolutionary Computation in Bioinformatics*, pages 279–294, Morgan Kaufmann Publishers, San Francisco, 2002.

[7] D.E. Goldberg. *Algorithm in Search, Optimization and Machine Learning*. Addison Wesley, Reading, USA, 1989.

[8] D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proc. 2nd International Conference on Genetic Algorithms*, pages 41–49, 1987.

[9] A.A. Kavalenka, B. Filipič, M.A. Hemminga, and J. Štrancar. Speeding up a genetic algorithm for EPR-based spin label characterization of biosystem complexity. *J. Chem. Inf. Model.*, 45(6):1628–1635, 2005.

[10] Q.Z. Kenny and L. Ziwei. Population diversity in permutation-based genetic algorithm. In *Proceedings of the 15th European Conference on Machine Learning (ECML 2004)*, pages 537–547, Pisa, Italy, 2004.

[11] J.P. Li, M.E. Balazs, G.T. Parks, and P.J. Clarkson. A species conserving genetic algorithm for multimodal function optimization. *Evol. Comput.*, 11(1):107–109, 2003.

[12] S.W. Mahfoud. Niching Methods for Genetic Algorithms. Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, 1995.

[13] A.A. Pétrowski. Clearing procedure as a niching method for genetic algorithms. In *Proc. IEEE International Conference on Evolutionary Computation (ICEC 1996)*, pages 798–803, Nagoya, Japan, 1996.

[14] W. Spears. Simple Subpopulation Schemes. In *Proc. 3rd Annual Conference on Evolutionary Programming*, pages 296–307, 1994.

[15] J. Štrancar, T. Koklič, Z. Arsov, B. Filipič, D. Stopar, and M.A. Hemminga. Spin Label EPR-based Characterization of Biosystem Complexity. *J. Chem. Inf. Model.*, 45(2):394–406, 2005.

[16] J. Štrancar, M. Šentjurc, and M. Schara. Fast and accurate characterization of biological membranes by EPR spectral simulations of nitroxides. *J. Magn. Reson.*, 142(2):254–265, 2000.

[17] F. Streichert, G. Stein, H. Ulmer, and A. Zell. A Clustering Based Niching Method for Evolutionary Algorithms. *Lect. Notes Comput. Sc.*, 2723:644–645, 2003.

DGPF – AN ADAPTABLE FRAMEWORK FOR DISTRIBUTED MULTI-OBJECTIVE SEARCH ALGORITHMS APPLIED TO THE GENETIC PROGRAMMING OF SENSOR NETWORKS

Thomas Weise, Kurt Geihs
Distributed Systems Group
University of Kassel, Germany
{weise,geihs}@vs.uni-kassel.de

Abstract We present DGPF, a framework providing multi-objective, auto-adaptive search algorithms with a focus on Genetic Programming. We first introduce a Common Search API, suitable to explore arbitrary problem spaces with different search algorithms. Using our implementation of Genetic Algorithms as an example, we elaborate on the distribution utilities of the framework which enable local, Master/Slave, Peer-To-Peer, and P2P/MS hybrid distributed search execution. We also discuss how heterogeneous searches consisting of multiple, cooperative search algorithms can be constructed. Sensor networks are distributed systems of nodes with scarce resources. We demonstrate how Genetic Programming based on our framework can be applied to create algorithms for sensor nodes that use these resources very efficiently.

Keywords: Auto-adaptation, Distributed genetic algorithms, Genetic programming, Heuristic, Randomized, Search algorithms, Sensor networks, Sensor nodes

1. Introduction

Find an election algorithm for a given sensor network with minimum instruction count which minimizes energy consumption due to transmissions. Construct the best aerodynamic shape of an airplane wing while maximizing its stability using the minimal amount of material. Many search algorithms can be applied to solve such complex problems [16]. There exist artificial approaches like Tabu Search or randomized Hill Climbing, physically inspired ones like Simulated Annealing as well as methods of biological origin like Genetic Algorithms.

For most problems it is not a priori possible to decide which algorithm and parameter configuration will perform best. Practical experiences often apply to a certain problem only and cannot be generalized. To implement different search algorithms or to customize multiple search libraries however is normally costly and time-consuming.

The performance of search algorithms can often be improved by distributing the computational load to a network of computers. If parameters like the mutation-rate in Genetic Algorithms or the length of the Tabu-List in Tabu Search are adapted dynamically, the performance may increase further [2].

When performing Genetic Programming of real algorithms with its usually very rugged fitness landscapes, a search framework taking advantage of all these improvement options is needed.

In this paper we introduce a Common Search API of our DGPF framework [7, 17], allowing the evaluation of arbitrary problem spaces to be performed with different or even multiple cooperating, distributed, auto-adaptive search algorithms. We will furthermore show the utility of our framework for automated algorithm creation for sensor networks by evaluating an experiment.

2. Related Work

In the past there have been successful applications of other search methodologies as back-ends of GA [9, 10, 14]. Meta-Heuristics like the one introduced by Bachelet and Talbi [1] already confirmed that the cooperation of different, hierarchical coupled search algorithms provides remarkable advantages. Yao has melted GA and Simulated Annealing together to create a new, improved version, the Genetic Annealing [20]. Our framework extends such ideas by integrating arbitrary search algorithms to cooperatively work together on one problem.

O'Reilly and Oppacher have suggested replacing GA as foundation for GP [13] with other heuristics like Simulated Annealing and Stochastic Iterated Hill Climbing. Applying such methods is simplified by our framework a lot. The GP layer or any other given Problem Space Implementation can rest on the Common Search API, which internally might run any search algorithm implemented.

Most of the research stated above does not concern multi-objective optimization [16]. Our search API on the other hand provides building blocks which ease the construction of such algorithms.

A lot of work has been done on the self-adaptation of search algorithms [2, 3]. If a search heuristic is implemented using our framework, it will automatically be equipped with this ability too. It may use different strategies that can even be exchanged at runtime.

3. Framework Structure

The core of our framework is formed by a Common Search API, which defines some classes and prototypes to be used. This API can be accessed from two sides (see Figure 1): On the one side different search algorithms can be implemented, providing the functionality needed to perform randomized heuristic searches. The user, on the other side, has to implement the functionality needed to explore the problem space and, if needed, to simulate possible solutions. In the style of multi-objective Genetic Algorithms, she may use different fitness functions to evaluate the simulated solution candidates. The user-defined code can then be used in conjunction with any search algorithm made available by the framework. Hence, a direct comparison and selection of the optimal approach for a given problem has become straightforward.

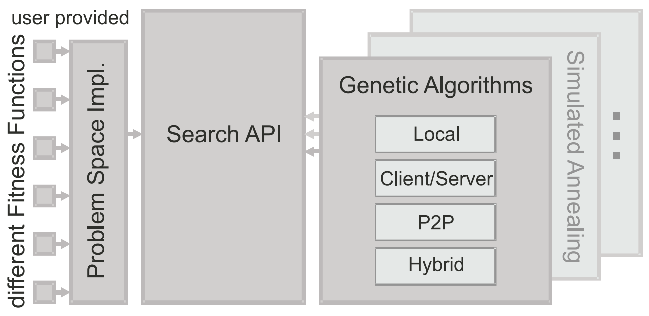


Figure 1. The structure of the DGP Framework's search abilities.

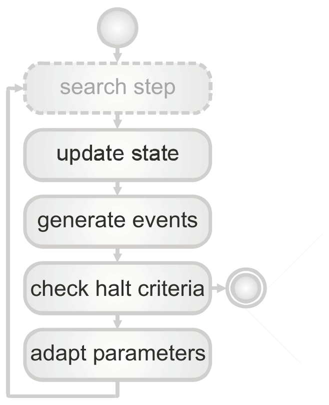


Figure 2. The control loop FSM.

3.1 The Common Search API

The search API introduces four essential tools and abstractions: a finite state machine which governs the control loop shared by the search algorithms, means for the user to plug in fitness functions and problem domain specific functionality, basic auto-adaptation support, and distribution utilities.

Search algorithms in general, if bio-inspired, randomized, or otherwise heuristic, can be performed using the finite state machine presented in Figure 2 as control loop. They can be divided into single search steps representing for example a generation in Genetic Algorithms, or a state transition in Simulated Annealing. After each step the state information, for instance containing the best individual found in the search and its fitness values, will be updated. Status events will now be generated to inform the application using the search algorithm. To limit the runtime of the search, the user may provide certain thresholds, like a maximum search time, a maximum update count, optimum fitness values, and so on, in order to define when the search should be halted automatically. If these criteria are not met, the search parameters can auto-adapt to the new situation and the next step will be initiated. To investigate a custom problem space, the user has to plug in the "Problem Space Implementation" (see Figure 1) which consists of three parts:

1 The type of individuals to be examined, which can be anything from simple numbers if optimizing a mathematical problem to complex construction plans for airplane wings.

2 The methods needed to randomly create initial individuals and to derive new individuals from either one or two already existing ones.

3 Means to simulate these individuals in order to check their fitness.

Based on this implementation the user can now define multiple fitness functions, regarding different functional and non-functional aspects of the individuals evaluated.

Apart from Genetic Programming for sensor networks, we exemplarily created a Problem Space Implementation for Semantic Web Service composition, able to solve problems like the WSC Challenge [19], as a proof of concept.

The Common Search API includes facilities for both parallelization and distribution which will be discussed in the next section.

3.2 Genetic Algorithms and the Distribution Schemes of the DGPF

The most popular biologically inspired search and optimization methods by far are Genetic Algorithms. Genetic Algorithms follow a well known schema which closely matches the search control loop FSM introduced in the previous section. Starting with an initial population, the individuals are evaluated, statistical information is updated and individuals are selected for reproduction in the next iteration.

Distributed Genetic Algorithms outclass their locally running counterparts in many applications [6]. Let us thus discuss the distribution utilities of the Common Search API exemplarily for the DGPF implementation of GA.

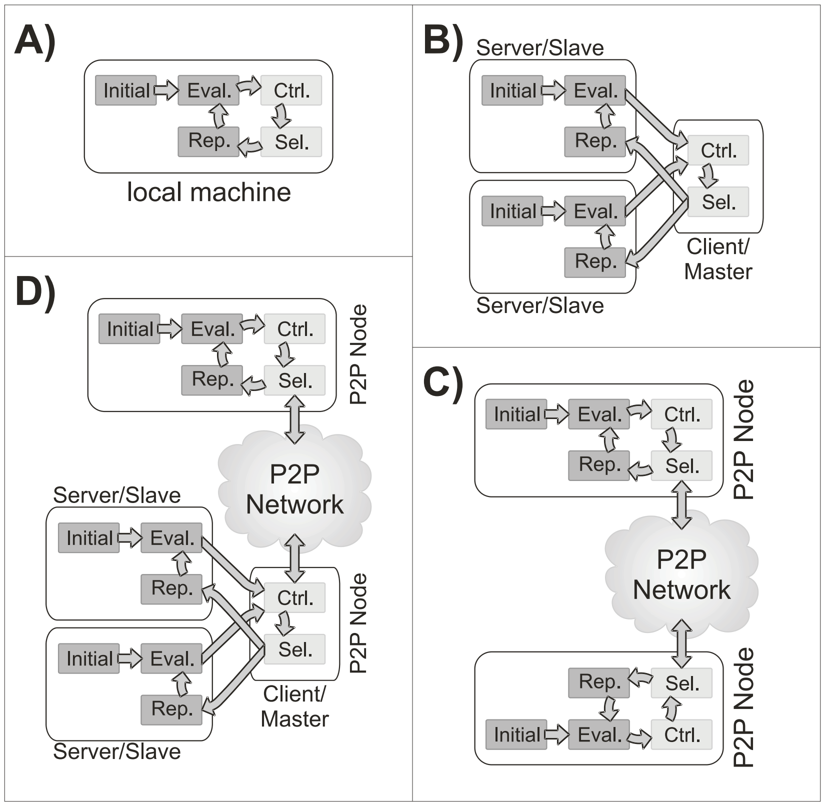


Figure 3. Different distribution schemes for GA provided by the DGPF.

Four different distribution forms of Genetic Algorithms are provided, as illustrated in Figure 3. The default method is to run a search locally (A). If more than one machine is available in a network, the tasks of creating and evaluating individuals can be distributed. This technique is called Client/Server or Master/Slave (B) approach [4, 11]. This is useful if the evaluation involves time-consuming simulations. If network bandwidth is limited or large populations are needed, a Peer-To-Peer approach should be chosen (C) [6, 12]. Different machines running Genetic Algorithms are now able to cooperate using the Island Hopping paradigm. Last but no least, a hybrid distribution scheme (D) of mixing the Peer-To-Peer and Client/Server techniques allows different networks or clusters to cooperate on the same search.

The Client/Server- and the P2P-components are unified in the Common Search API. Therefore, they can be used by all search algorithms implemented in the DGPF, allowing even totally different algorithms like GA, Simulated

Annealing, Tabu Search and Hill Climbing to be incorporated into one heterogeneous search.

The distribution methods discussed here are built using self-healing and error-tolerant techniques. Thus, a Client/Server system will continue its work even if all but one server are switched off by a hardware-reset. A search using the P2P-distribution will keep running even if all other P2P-nodes are shut down. If some of the other machines happen to be restarted, they will seamlessly be integrated into the search again by both technologies.

Other unified base structures of the DGPF are comparators, sorting and selection schemes. A comparator is used to determine which individuals are dominated by which other ones. The sorting schemes allow individuals to be sorted according to these comparators or by using additional statistical measures. Combined with the non-dominated individual list maintained by the API, multi-objective search algorithms can easily be constructed. As a combination of these four features, the NPG-Algorithm [8] has exemplarily been implemented.

4. Genetic Programming of Sensor Networks

Today we experience a growing demand for distributed systems of sensors [5]. In this chapter, we describe how the DGPF framework is used to genetically create algorithms for such sensor networks.

Sensor nodes are small devices that gather sensor information about their environment and transmit it wirelessly. They are restricted in resources like memory size, processing speed, and, most important, battery power. The communication among them is not reliable and the topology of their network is volatile. The program code created for sensor nodes should thus be robust and as efficient as possible.

Our goal is the automated creation of algorithms for sensor nodes. We apply multi-objective Genetic Programming since it allows optimizing the algorithms created not only for functionality but also for the economical use of resources, especially for minimizing expensive communication.

To evaluate the fitness of such algorithms we simulate whole sensor networks. In our model, sensor nodes are represented as virtual machines with a fixed-sized memory architecture, asynchronous IO, and a Turing-complete instruction set [15, 18].

Many nodes (the virtual machines) run asynchronously in the simulation at approximately the same speed which, however, might differ from node to node and cannot be regarded as constant. The nodes are connected wirelessly and thus cannot a priori guarantee reliable communication. It is not possible to send directed transmissions. Like radio broadcasts they will be received by any node in range.

With such simulations we can transform global behavior of a network into local behavior of single nodes using GP.

4.1 Testing the Features of the DGPF for GP

To validate the utility of our framework for genetically programming sensor networks we chose an example problem well known in the area of distributed systems: the election. Election means to select one node out of a group of nodes, to act as communication relay, for instance. All nodes should receive knowledge of the id of this special node. One way to perform such an election would be to determine the maximum id of all nodes.

In order to solve this problem, we initialize all automata with their own id in the first memory cell. If an algorithm makes progress at all, the nodes should have stored greater (valid) ids there after some time. A fully functional algorithm would accomplish that the first memory cells of all nodes contain the maximum id. If the algorithm is also resource-friendly, it should reach this goal needing as few transmissions as possible.

Therefore we apply three fitness functions: the first function is the cumulative of all valid ids stored in the first memory cells of the nodes in all time steps (*i*), see Figure 4. It is therefore an indicator both for the functionality as well as the convergence speed of the algorithms. The second fitness function is inversely proportional to the count of messages sent by all nodes (*ii*) and the third function is inversely proportional to the instruction count of the algorithms found (*iii*).

As experimental setting we use six normal PCs in a network to perform a) homogeneously distributed, non-adaptive GA using the P2P-scheme described in Section 3.2 as well as *b*) randomly configured adaptive heterogeneous searches (also P2P distributed). For the experiments of type *a*), four different population sizes are tested: 2048, 4096, 6144 and 8192. When performing the experiments of the second type, each node picks a search algorithm (GA, Hill Climbing, Simulated Annealing). If using GA it chooses a selection scheme (e.g., Tournament Selection), picks a population size ($\leq$ 8192), determines mutation/crossover rates, configures the caches and such and such all randomly. For each experiment, a fixed runtime of two hours is granted. The two experiments are repeated eight times each. In Figure 4 we have plotted the fitness values of the non-dominated algorithms found by both approaches during all runs, leaving away those algorithms having minimal code size or minimal transmissions while having no functional effect at all.

It now becomes clear that the auto-adaptive, randomly configured experimental setting, which takes (by chance) full advantage of all features of the DGPF, is able to find more algorithms with good functionality than a standard approach would yield. Both methods whatsoever were able to find working solutions for the election problem.

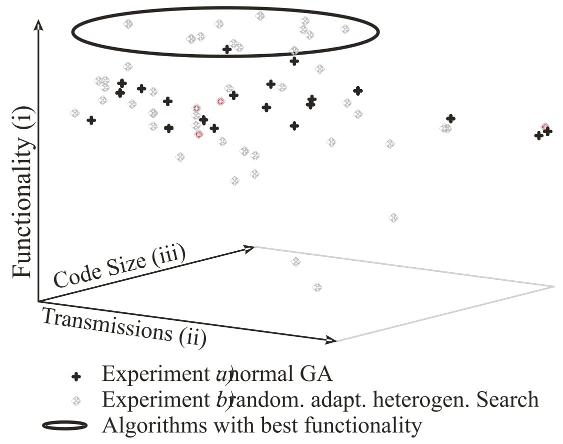


Figure 4. Utility of the DGPF-Features for GP.

A trivial (and thus more understandable) one of these solutions is displayed in Figure 5. The algorithm consists of two parts: a procedure called when the node starts up (procedure_0) and an asynchronously called, interrupt-like routine which receives incoming messages (procedure_1). In this simple algorithm, the nodes constantly broadcast the greatest id they encountered in a loop, reducing network traffic only by performing dummy work. In Figure 4 this program is represented by a gray dot in the left of the black ellipsis.

```
             called on startup   procedure_0
       store 1st variable into    0: push mem[0]
                 output buffer    1: some useless operations used
                                  2: to stall and, as a consequence,
                                  3: reduce transmissions in the
                                  4: simulated/evaluated time span
            send output buffer    5: send
              go back to start    6: goto 0
---------------------------------------------------------------
    called asynchronously when
           a message comes in    procedure_1
        compare the known and
           the received value     0: zf = (mem[-1] < mem[0])
  if no improvement then exit     1: if zf then goto 3 // ~~exit~~
              exchange values     2: xchg mem[-1], mem[0]
```

Figure 5. One of the non-dominated solutions found.

5. Future Work and Conclusion

There are three tasks in our research which are currently in progress and soon to be completed. The first one is the integration of additional bio-inspired search algorithms into the DGPF and the evaluation of their utility for Genetic Programming.

We will soon be able to provide an easy-to-use control and monitoring interface for the DGPF. It will graphically present the rich statistical information collected from the events created by the control FSMs. The user will be able to control a distributed search, to modify all parameters of the different nodes manually and to access the search results at any given time.

The focus of our development effort is put on Genetic Programming and its application to sensor networks. We are now able to perform research on different technologies in this area since we have laid a solid foundation of efficient search algorithms suitable for this purpose. With this foundation and the results of our future research, we hope to increase the performance of Genetic Programming and the quality of its results in that sector significantly.

In this paper we have presented a framework for heuristic randomized multi-objective search algorithms that incorporates the results of many of the best contributions to the area of randomized heuristic search. Although our own research interests concentrate on Genetic Programming, our new search API can easily be customized to any given problem space. The resulting auto-adaptive applications can be distributed over a network, performing heterogeneous, co-operative searches. Furthermore, we provide the framework and all results to the research community under the LGPL. More information on our research as well as the fully documented Java source code of the DGPF can be found at `http://dgpf.sourceforge.net` [7].

References

[1] V. Bachelet and E.-G. Talbi. A Parallel Co-evolutionary Metaheuristic. *Lect. Notes Comput. Sc.*, 1800:628–635, 2000.

[2] T. Bäck and M. Schütz. Intelligent Mutation Rate Control in Canonical Genetic Algorithms. *Lect. Notes Comp. Sc.*, 1079:158–167, 1996.

[3] D. Büche, S.D. Müller, and P. Koumoutsakos. Self-Adaptation for Multi-objective Evolutionary Algorithms. In *Proc. Second International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, Faro, Portugal, 2003.

[4] E. Cantu-Paz. Designing Efficient Master-Slave Parallel Genetic Algorithms. In *Proc. Third Annual Conference on Genetic Programming*, 1998.

[5] C.-Y. Chong and S.P. Kumar. Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE* 91(8):1247–1256, 2003.

[6] F.S. Chong and W.B. Langdon. Java based Distributed Genetic Programming on the Internet. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 1999)*, Volume 2, page 1229, Orlando, FL, USA, 1999.

[7] Distributed Genetic Programming Framework, LGPL licensed, Open-Source Java Framework. `http://dgpf.sourceforge.net`.

[8] J. Horn, N. Nafpliotis, and D. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proc. First IEEE Conference on Evolutionary Compuation*, Volume 1, pages 82–87, Orlando, FL, USA, 1994.

[9] H. Ishibuchi, T. Yoshida, and T Murata. Balance between genetic search and local search in hybrid evolutionary multi-criterion optimization algorithms. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2002)*, New York, NJ, USA, 2002.

[10] D. Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers & Operations Research*, 23(6):547-558, 1996.

[11] S. Luke. ECJ: A Java-based evolutionary computation and genetic programming system, 2000. `http://cs.gmu.edu/~eclab/projects/ecj/`.

[12] W.N. Martin, J. Lienig, and J.P. Cohoon. Island (migration) models: evolutionary algorithms based on punctuated equilibria. In Bäck, Fogel, Michalewicz (eds.), *Handbook of evolutionary Computation*, IOP Publishing and Oxford University Press, 1997.

[13] U.-M. O'Reilly and F. Oppacher. Program Search with a Hierarchical Variable Length Representation: Genetic Programming, Simulated Annealing and Hill Climbing. *Lect. Notes Comput. Sc.*, 866:397–406, 1994.

[14] L. Shi and S. Olafsson. A New Hybrid Genetic Algorithm. In *Late Breaking Papers at the Genetic Programming 1998 Conference*, Madison, WI, USA, 1998.

[15] A. Teller. Turing completeness in the language of genetic programming with indexed memory. In *Proc. IEEE World Congress on Computational Intelligence*, Volume 1, Orlando, FL, USA, 1994.

[16] M. Villalobos-Arias, C.A . Coello Coello, and O. Hernández-Lerma. Asymptotic Convergence of Some Metaheuristics Used for Multiobjective Optimization. *Lect. Notes Comput. Sc.*, 3469:95–111, 2005.

[17] T. Weise and K. Geihs. Genetic Programming Techniques for Sensor Networks. In *Proc. 5. GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze"*, Stuttgart, Germany, 2006.

[18] J.R. Woodward. Evolving turing complete representations. In *Proc. Congress on Evolutionary Computation (CEC 2003)*, Volume 2, pages 830–837, Birmingham, UK, 2003.

[19] WS-Challenge 06. `http://insel.flp.cs.tu-berlin.de/wsc06/`.

[20] X. Yao. Optimization by genetic annealing. In *Proc. Second Australian Conference on Neural Networks*, pages, 94–97, Sydney, Australia, 1991.

COMPUTER-ASSISTED ANALYSIS OF A METALLURGICAL PRODUCTION PROCESS IN VIEW OF MULTIPLE OBJECTIVES

Bogdan Filipič, Tea Tušar
Department of Intelligent Systems
Jožef Stefan Institute, Ljubljana, Slovenia
{bogdan.filipic,tea.tusar}@ijs.si

Erkki Laitinen
Department of Mathematical Sciences
University of Oulu, Finland
erkki.laitinen@oulu.fi

Abstract Numerical experiments in optimizing secondary coolant flows on a steel casting machine with respect to multiple objectives were performed using the recently proposed Differential Evolution for Multiobjective Optimization (DEMO). Calculations were done for a selected steel grade under the assumption of steady-state caster operation. Their aim was to find suitable sets of coolant flow settings under conflicting requirements for minimum temperature deviations and predefined core length in the caster. In contrast to solutions produced in single-objective optimization, approximation sets of Pareto optimal fronts provide more information to a plant engineer and allow for better insight into the casting process behavior.

Keywords: Continuous casting of steel, Coolant flows, DEMO, Differential evolution, Multiobjective optimization, Process parameters

1. Introduction

Like the majority of modern production processes, material production and processing nowadays strongly rely on numerical analysis and computer support. Numerical simulators enable insight into process development, allow for execution of numerical experiments and facilitate manual process optimization. Moreover, reliable process simulators and efficient optimization procedures make it possible to automate process parameter optimization and improve material properties. A way of achieving these goals is to couple the process simulator with an optimization algorithm via a cost function which allows for automatic assessment of the simulation results.

Continuous casting of steel is an example of a process to which novel computational approaches have been applied intensively over the last years to enhance product characteristics and minimize production costs. In this complex metallurgical process molten steel is cooled and shaped into semi-manufactures. To cast high quality steel, it is important to properly control the metal flow and heat transfer during the process. They depend on numerous parameters, including the casting temperature, casting speed and coolant flows. Finding optimal values of process parameters is hard since various, often conflicting criteria need to be applied, the number of possible parameter settings is high, and parameter tuning through real-world experimentation is not feasible because of safety risk and high costs. Techniques applied to overcome these difficulties include knowledge-based heuristic search [2] and evolutionary algorithms [1, 6, 8, 9]. However, the predominant optimization approach taken in the applied studies so far was to aggregate multiple criteria into a single cost value and solve the optimization problem empirically using the simulator-optimizer coupling.

In this paper we report on preliminary numerical experiments in optimizing secondary coolant flows on a steel casting machine with respect to multiple objectives using a multiobjective optimization evolutionary algorithm. Calculations were done for a selected steel grade under the assumption of steady-state caster operation. Their purpose was to get better insight into process behavior and find optimized sets of coolant flow settings under conflicting objectives. The paper describes the optimization task and the multiobjective optimization approach, and reports on the performed numerical experiments and obtained results.

2. The Optimization Task

In continuous casting, liquid steel is poured into a bottomless mold which is cooled with internal water flow. The cooling in the mold extracts heat from the molten steel and initiates the formation of a solid shell. The shell formation is crucial for the support of the slab behind the mold exit. The slab then enters into the secondary cooling area in which it is cooled by water sprays. The secondary cooling region is divided into cooling zones where the amount of the cooling water can be controlled separately.

In this study we consider a casting machine with the secondary cooling area divided into nine zones. In each zone, cooling water is dispersed to the slab at the center and corner positions. Target temperatures are specified for the slab center and corner in every zone. Water flows should be tuned in such a way that the resulting slab surface temperatures match the target temperatures as closely as possible. From metallurgical practice this is known to reduce cracks and inhomogeneities in the structure of the cast steel. Formally, cost function c_1 is introduced to measure deviations of actual temperatures from the target ones:

$$c_1 = \sum_{i=1}^{N_Z} |T_i^{\text{center}} - T_i^{\text{center}*}| + \sum_{i=1}^{N_Z} |T_i^{\text{corner}} - T_i^{\text{corner}*}|, \tag{1}$$

where N_z denotes the number of zones, T_i^{center} and T_i^{corner} the slab center and corner temperatures in zone i, and $T_i^{\text{center}*}$ and $T_i^{\text{corner}*}$ the respective target temperatures in zone i.

There is also a requirement for core length, l^{core}, which is the distance between the mold exit and the point of complete solidification of the slab. The target value for the core length, $l^{\text{core}*}$, is prespecified, and the actual core length should be as close to it as possible. Shorter core length may result in unwanted deformations of the slab as it solidifies to early, while longer core length may threaten the process safety. We formally treat this requirement as cost function c_2:

$$c_2 = |l^{\text{core}} - l^{\text{core}*}|. \tag{2}$$

The optimization task is to minimize both c_1 and c_1 over possible cooling patterns (water flow settings). It is known that the two objectives are conflicting, hence it is reasonable to handle this optimization problem as a multiobjective one. Water flows cannot be set arbitrarily, but according to the technological constraints. For each zone, minimum and maximum values are prescribed for the center and corner water flows.

A prerequisite for optimization of this process is an accurate numerical simulator, capable of calculating the temperature field in the slab as a function of process parameters and evaluating it with respect to cost functions Eqn. (1) and Egn. (2). For this purpose we used the mathematical model of the process with Finite Element Method (FEM) discretization of the temperature field and the corresponding nonlinear equations solved with relaxation iterative methods, already applied in previous single-objective optimization study of the casting process [7].

3. Multiobjective Optimization

3.1 Pareto Optimality

Consider the multiobjective optimization problem (MOP) of finding the minimum of the cost function $\mathbf{c}$:

$$\begin{aligned} \mathbf{c}&\colon X \to Z \\ \mathbf{c}&\colon (x_1, \ldots, x_n) \mapsto (c_1(x_1, \ldots, x_n), \ldots, c_m(x_1, \ldots, x_n)), \end{aligned}$$

where X is an n-dimensional decision space, and $Z \subseteq \mathbb{R}^m$ is an m-dimensional objective space ($m \geq 2$). The objective vectors from Z can be partially ordered using the concept of *Pareto dominance*: $\mathbf{z}^1$ *dominates* $\mathbf{z}^2$ ($\mathbf{z}^1 \prec \mathbf{z}^2$) iff $\mathbf{z}^1$ is not worse than $\mathbf{z}^2$ in all objectives and better in at least one objective. When the objectives are conflicting, there exists a set of optimal objective vectors called *Pareto optimal front*. Each vector from the Pareto optimal front represents a different trade-off between the objectives and without additional information no vector can be preferred to another.

With a multiobjective optimizer we search for an *approximation set* that approximates the Pareto optimal front as well as possible. When solving MOPs in practice it is often important to provide the user with a diverse choice of trade-offs. Therefore, beside including vectors close to the Pareto optimal front, the approximation set should also contain near-optimal vectors that are as distinct as possible.

3.2 DEMO

Finding a good approximation set in a single run requires a population-based method. Consequently, evolutionary algorithms have been frequently used as multiobjective optimizers [3]. Among them, the recently proposed Differential Evolution for Multiobjective Optimization (DEMO) [11] is applied in optimizing the described metallurgical process.

DEMO is based on Differential Evolution (DE) [10], an evolutionary algorithm for single-objective optimization that has proved to be very successful in solving numerical optimization problems. In DE, each solution is encoded as an n-dimensional vector. New solutions, also called candidates, are constructed using operations such as vector addition and scalar multiplication. After the creation of a candidate, the candidate is compared with its parent and the best of them remains in the population, while the other one is discarded.

Because the objective space in MOPs is multidimensional, DE needs to be modified to deal with multiple objectives. DEMO is a modification of DE with a particular mechanism for deciding which solution should remain in the population. For each parent in the population, DEMO constructs the candidate solution using DE. If the candidate dominates the parent, it replaces the parent in the current population. If the parent dominates the candidate, the candidate is discarded. Otherwise, if the candidate and its parent are incomparable, the candidate is added to the population. After constructing candidates for each parent individual in the population, the population has possibly increased. In this case, it is truncated to the original size using nondominated sorting and crowding distance metric (as in NSGA-II [4]). This steps are repeated until a stopping criterion is met.

DEMO is a simple but powerful algorithm, fully presented in [11] in three variants. Throughout this paper, the elementary variant, called DEMO/parent, is used.

4. Experiments and Results

4.1 Experimental Setup

Numerical experiments in multiobjective optimization of the casting process were performed for a selected steel grade with the slab cross-section of 1.70 m $\times$ 0.21 m. Candidate solutions were encoded as 18-dimensional real-valued vectors, representing water flow values at the center and corner positions in 9 zones of the secondary cooling area. Search intervals for cooling water flows at both center and corner positions in zones 1, 2 and 3 were between 0 and 50 m^3/h, while in the zones 4–9 between 0 and 10 m^3/h. Table 1 shows the prescribed target slab surface temperatures. The target value for the core length $l^{\mathrm{core}*}$ was 27 m.

Table 1. Target surface temperatures in $^\circ$C.

Zone number	1	2	3	4	5	6	7	8	9
Center position	1,050	1,040	980	970	960	950	940	930	920
Corner position	880	870	810	800	790	780	770	760	750

DEMO was integrated with the numerical simulator of the casting process into an automated optimization environment. DEMO evolved sets of candidate solutions in search for a good approximation set, and the simulator served as a solution evaluator. Steady-state operation of the casting machine was assumed and optimization performed in the off-line manner.

The most limiting factor for experimental analysis is the computational complexity of the casting process simulation. A single simulator run takes about 40 seconds on a 1.8-GHz Pentium IV computer. In initial experimentation we found DEMO runs with 5,000 solution evaluations (and therefore taking about 55 hours) well compromising between the execution time and solution quality. Further algorithm settings were also adopted according to the initial parameter tuning experiments [5] and were as follows: population size 50, number of generations 100, scaling factor 0.5 and crossover probability 0.05.

4.2 Results and Discussion

The primary result of this study were approximation sets of Pareto optimal fronts. Figure 1 shows the approximation sets found by DEMO for five casting speeds, ranging from 1.0 m/min to 1.8 m/min. Each set of nondominated solutions is the final result of a single DEMO run at a constant casting speed.

We can observe that the two objectives are really conflicting in the sense that finding a minimum for one of them the optimization procedure fails to do so for the other and vice versa. It is also obvious that the casting speed has a

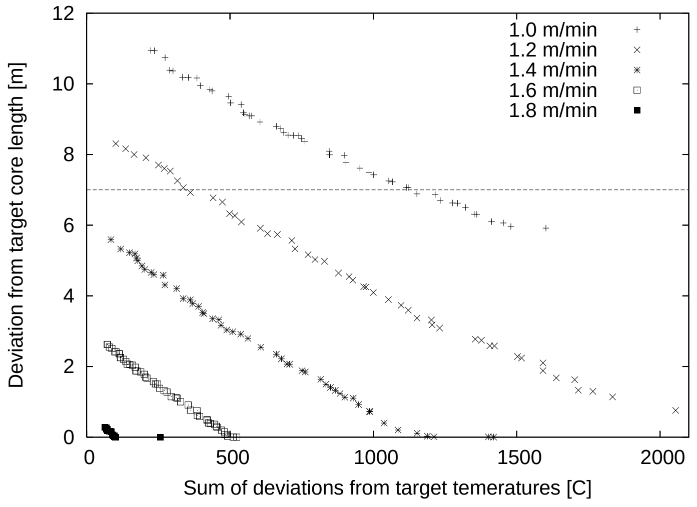


Figure 1. Nondominated solutions found with DEMO for different casting speeds. The dashed horizontal line denotes the maximum allowed deviation of the core length from the target value (7 m).

decisive impact on the result. Moreover, the higher the casting speed, the more the two objectives can be met simultaneously. This corresponds with practical experience on the considered casting machine, where the process is easier to control at the usual casting speed (1.6–1.8 m/min). Lower casting speed is clearly shown as disadvantageous and in practice it is only set exceptionally, for example, when a new batch of steel is awaited.

A detailed analysis of the solution properties also reveals that, in view of the objective c_1, the majority of actual surface temperatures are higher than the target temperatures, while regarding c_2, the actual core length is almost always shorter than the target value. Unexpectedly, the deviation is sometimes even greater than 7 m, meaning that the actual core length is less than 20 m, which is unacceptable. This threshold value is shown in Figure 1 and should be considered as an additional constraint in future studies.

Looking into decision space, one can also observe certain rules. In case of applying trade-off solutions from the middle of the approximation sets, the amount of coolant spent increases with the casting speed (see the left-hand side diagrams in Figures 2–6). This is an expected result as higher casting speed implies more intense cooling. On the other hand, the distributions of temperature differences across the secondary cooling zones (right-hand side diagrams in Figures 2–6) exhibit two characteristics. First, the target temperatures are much more difficult to achieve at the center than in the corner slab positions. Second, the differences at the center are rather non-uniform. While some are close to zero, others reach up to 200 °C at lower casting speeds. Such a situation is not wanted in practice calls for reformulation of objective c_1.

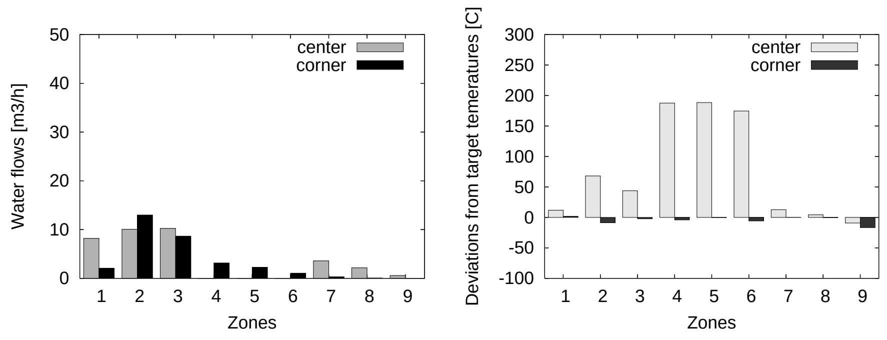


Figure 2. A trade-off solution from the middle of the approximation set for the casting speed speed of 1.0 m/min: c_1 = 740 °C, c_2 = 8.5 m.

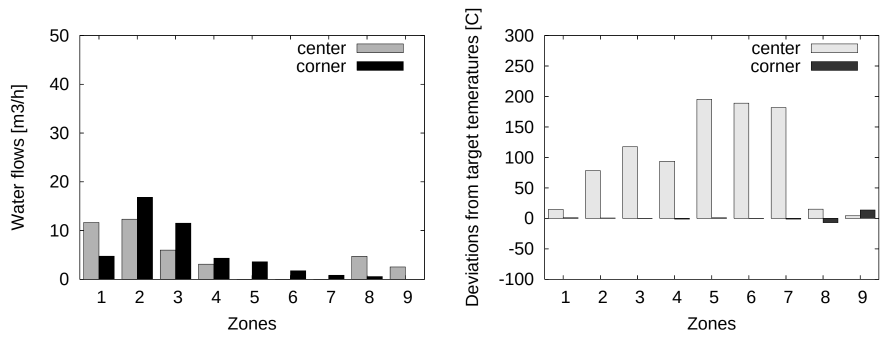


Figure 3. A trade-off solution from the middle of the approximation set for the casting speed speed of 1.2 m/min: c_1 = 915 °C, c_2 = 4.5 m.

Finally, it is worth checking the extreme solutions from an approximation set at a given casting speed. Figures 7 and 8 clearly show how one objective is met at the expense of the other. None of these would normally be used in practice. Instead, a plant engineer would rather select a trade-off setting balancing between the two objectives.

5. Conclusion

Optimization of process parameter settings in continuous casting of steel is a key to higher product quality. Nowadays it is often performed through virtual experimentation involving numerical process simulators and advanced optimization techniques. In this preliminary study of optimizing 18 cooling

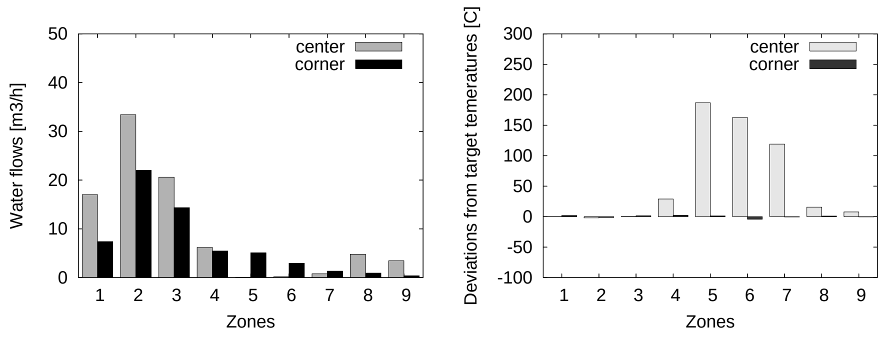


Figure 4. A trade-off solution from the middle of the approximation set for the casting speed speed of 1.4 m/min: c_1 = 537 °C, c_2 = 2.9 m.

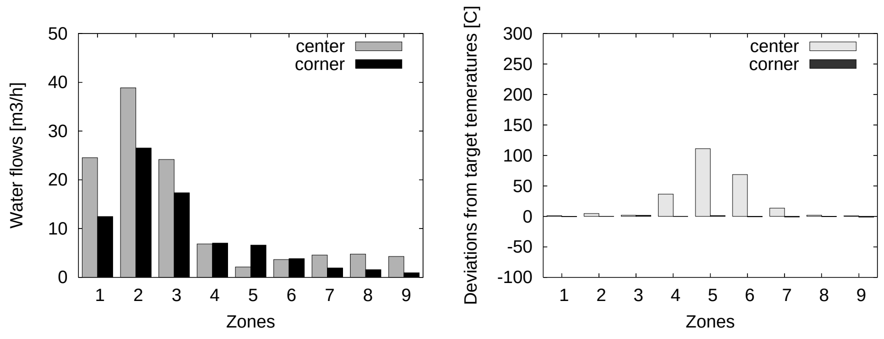


Figure 5. A trade-off solution from the middle of the approximation set for the casting speed speed of 1.6 m/min: c_1 = 247 °C, c_2 = 1.5 m.

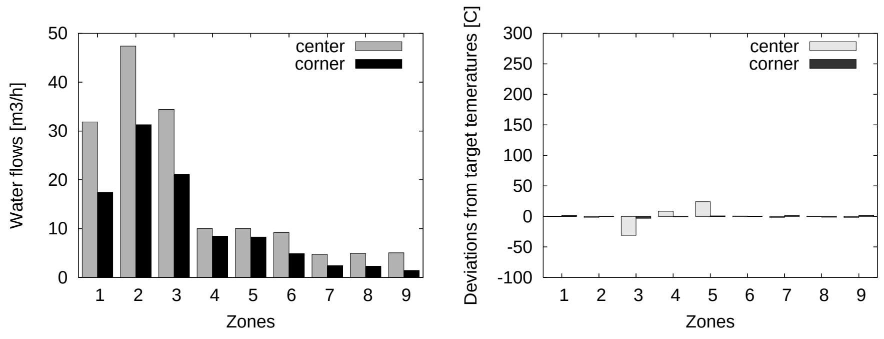


Figure 6. A trade-off solution from the middle of the approximation set for the casting speed speed of 1.8 m/min: c_1 = 80 °C, c_2 = 0.2 m.

water flows for an industrial casting machine the multiobjective optimization was brought into play.

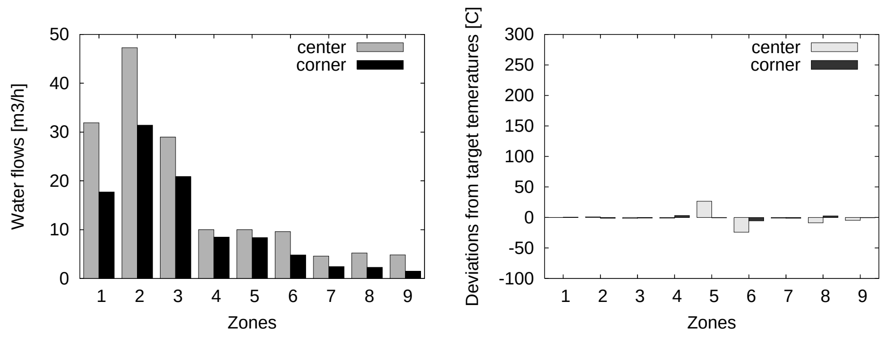


Figure 7. The leftmost solution from the approximation set for the casting speed speed of 1.4 m/min: c_1 = 85 °C, c_2 = 5.6 m.

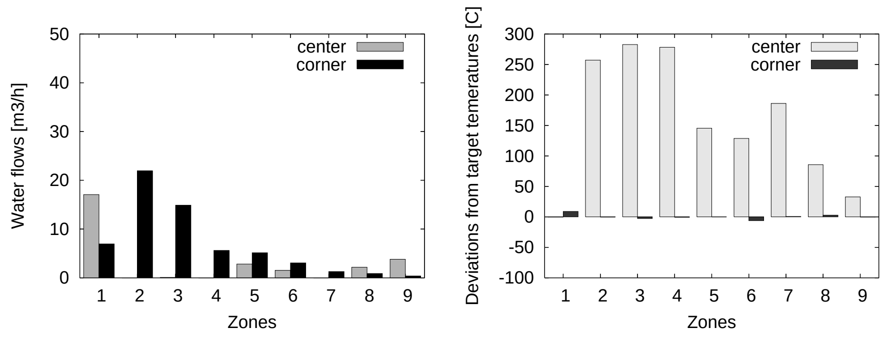


Figure 8. The rightmost solution from the approximation set for the casting speed speed of 1.4 m/min: c_1 = 1,419 °C, c_2 = 0.0 m.

The analysis assumes steady-state process conditions, hence the results are not primarily intended for control purposes but rather for better understanding of the process and evaluation of the casting machine performance. The resulting approximation sets of Pareto optimal fronts indeed offer a more general view of the process properties. The results support some facts already known in practice and, at the same time, show critical points, such as the need to reformulate the temperature deviation criterion to ensure uniform distribution of temperature differences over the zones, and extend the optimization problem definition with an additional constraint. From the practical point of view, further studies will also explore how much the optimization results are affected by the factors that were kept constant so far, such as steel grade, slab geometry and casting machine characteristics.

Acknowledgment

The work presented in the paper was supported by the Slovenian Research Agency and the Academy of Finland under the Slovenian-Finnish project BI-FI/04-05-009 *Numerical Optimization of Continuous Casting of Steel*, and by the Slovenian Research Agency under the Research Programme P2-0209 *Artificial Intelligence and Intelligent Systems*.

References

[1] N. Chakraborti, R. S. P. Gupta, and T. K. Tiwari. Optimisation of continuous casting process using genetic algorithms: studies of spray and radiation cooling regions. *Ironmaking and Steelmaking*, 30(4):273–278, 2003.

[2] N. Cheung and A. Garcia. The use of a heuristic search technique for the optimization of quality of steel billets produced by continuous casting. *Engineering Applications of Artificial Intelligence*, 14(2):229–238, 2001.

[3] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK, 2001.

[4] K. Deb, A. Pratap, S. Agrawal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA–II. *IEEE Trans. Evol. Comput.*, 6(2):182–197, 2002.

[5] M. Depolli, T. Tušar, and B. Filipič. Tuning parameters of a multiobjective optimization evolutionary algorithm on an industrial problem. In *Proc. Fifteenth International Electrotechnical and Computer Science Conference (ERK 2006)*, vol. B, pages 95–98, Portorož, Slovenia, 2006. In Slovenian.

[6] B. Filipič. Efficient simulation-based optimization of process parameters in continuous casting of steel. In D. Büche, N. Hofmann (Eds.), *COST 526: Automatic Process Optimization in Materials Technology: First Invited Conference*, pages 193–198, Morschach, Switzerland, 2005.

[7] B. Filipič and E. Laitinen. Model-based tuning of process parameters for steady-state steel casting. *Informatica*, 29(4):491-–496, 2005.

[8] B. Filipič and T. Robič. A comparative study of coolant flow optimization on a steel casting machine. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2004)*, vol. 1, pages 569–573, Portland, OR, USA, 2004.

[9] B. Filipič and B. Šarler. Evolving parameter settings for continuous casting of steel. In *Proc. 6th European Conference on Intelligent Techniques and Soft Computing (EUFIT'98)*, vol. 1, pages 444–449, Aachen, Germany, 1998.

[10] K. V. Price and R. Storn. Differential evolution – a simple evolution strategy for fast optimization. *Dr. Dobb's Journal*, 22(4):18–24, 1997.

[11] T. Robič and B. Filipič. DEMO: Differential evolution for multiobjective optimization. *Lect. Notes Comput. Sc.*, 3410:520–533, 2005.