Information Sciences 258 (2014) 140–159

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Adaptive generalized crowding for genetic algorithms

Ole J. Mengshoel [a], Severino F. Galán [b,*], Antonio de Dios [b]

[a] Carnegie Mellon University, NASA Ames Research Park, Moffett Field, CA 94035, USA
[b] Artificial Intelligence Dept., UNED, Juan del Rosal 16, 28040 Madrid, Spain

ARTICLE INFO

Article history:
Received 23 March 2013
Received in revised form 27 July 2013
Accepted 24 August 2013
Available online 4 September 2013

Keywords:
Genetic algorithms
Premature convergence
Generalized crowding
Scaling factor control

ABSTRACT

The genetic algorithm technique known as crowding preserves population diversity by pairing each offspring with a similar individual in the current population (pairing phase) and deciding which of the two will survive (replacement phase). The replacement phase of crowding is usually carried out through deterministic or probabilistic crowding, which have the limitations that they apply the same selective pressure regardless of the problem being solved and the stage of genetic algorithm search. The recently developed generalized crowding approach introduces a scaling factor in the replacement phase, thus generalizing and potentially overcoming the limitations of both deterministic and probabilistic crowding. A key problem not previously addressed, however, is how the scaling factor should be adapted during the search process in order to effectively obtain optimal or near-optimal solutions. The present work investigates this problem by developing and evaluating two methods for adapting, during search, the scaling factor. We call these two methods diversity-adaptive and self-adaptive generalized crowding respectively. Whereas the former method adapts the scaling factor according to the population's diversity, the latter method includes the scaling factor in the chromosome for self-adaptation. Our experiments with real function optimization, Bayesian network inference, and the Traveling Salesman Problem show that both diversity-adaptive and self-adaptive generalized crowding are consistent techniques that produce strong results, often outperforming traditional generalized crowding.

1. Introduction

Genetic algorithms (GAs) [19,16,11] are stochastic search methods based on natural evolution. They have been successfully applied to optimization problems in numerous fields. The processes of variation (recombination and mutation) and selection form the basis of GAs; whereas the former facilitates diversity and novelty, the latter favors quality. Generally, a GA's goal is to find an optimal (or near-optimal) solution, but this goal can be extended to identifying a set of diverse high-fitness solutions. In both cases, premature convergence to local but poor optima is a difficulty that frequently arises when applying GAs to complex problems. It is associated with the loss of population diversity; however, too much diversity can dramatically reduce GA efficiency. Therefore, the trade-off between exploitation of the best individuals and exploration of alternative search space regions is important in GA research and practice.

Crowding was introduced by de Jong [9] as a technique for preserving population diversity and preventing premature convergence. Crowding is applied in the survivor selection step of GAs in order to decide which individuals, among those in the current population and their offspring, will pass onto the next generation. It consists of two main phases: pairing

* Corresponding author. Tel.: +34 91 3987300; fax: +34 91 3989382.
E-mail addresses: ole.mengshoel@sv.cmu.edu (O.J. Mengshoel), seve@dia.uned.es (S.F. Galán), adedios@dia.uned.es (A. de Dios).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.08.056

and replacement [30]. In the pairing phase, offspring individuals are paired with individuals in the current population according to a similarity metric. In the replacement phase, a decision is made for each pair of individuals as to which of them will remain in the population.

Depending on how the replacement phase is carried out, there are two main types of crowding: deterministic [25,26] and probabilistic [29,28]. Whereas deterministic crowding selects the fittest individual in each pair in the replacement phase, probabilistic crowding selects the surviving individual for each pair based on a probabilistic formula that takes fitness into account. The design of the replacement rule has great impact on the performance of crowding. On the one hand, deterministic crowding develops an exploitative replacement strategy. This may be a disadvantage, since exploitative methods in GAs frequently lead to premature convergence. On the other hand, probabilistic crowding promotes the exploration of alternative (less fit) solutions. In probabilistic crowding, the degree of exploration is defined through a probabilistic formula that is not changed during GA execution. However, a more careful trade-off between exploration and exploitation can perhaps be achieved if selective pressure is adapted to the problem being solved and to the GA stage. These observations motivated the introduction of *generalized crowding* [14]. Generalized crowding is inspired by probabilistic crowding, however the degree of exploration can be controlled by means of a parameter named *scaling factor*. It turns out that both deterministic and probabilistic crowding are special cases of generalized crowding, obtained by assigning the scaling factor the values 0 and 1 respectively.

Since the introduction of generalized crowding [14], it has only been applied by keeping the scaling factor fixed throughout the GA generations. As the scaling factor parameter determines the applied selective pressure, the present work investigates how this parameter can be varied during the search process in order to effectively obtain optimal or near-optimal solutions. For example, more exploration in the first generations of the GA and more exploitation in the final generations usually produces a more effective search [10, Section 3]; simulated annealing is another search technique that takes advantage of this fact through a cooling schedule [22,1]. We explore in this work two novel approaches to automatically changing, or adapting, the scaling factor. Our methods still require an initial value for the scaling factor, however it is now changed during a GA's run and thus the impact of using a poor initial value is reduced. In particular, the novel diversity-adaptive and self-adaptive control techniques rely entirely on the GA's population to determine how the scaling factor is changed. In the diversity-adaptive method, the scaling factor is adapted according to the population's diversity. In the self-adaptive method, the scaling factor is adapted by including it in the chromosome. The impact of the scaling factor's initial value, as provided by the user, is much smaller for both these adaptive variants of generalized crowding compared to the fixed scaling factor used in traditional generalized crowding. As a consequence, the user can be much less concerned about how the scaling factor is initialized. Our experiments with GAs applied to real function optimization, Bayesian network inference, and the Traveling Salesman Problem show that diversity-adaptive and self-adaptive generalized crowding often produce better results, in a more consistent way, compared to generalized crowding using a fixed scaling factor. We perform ANOVA tests that confirm the statistical significance of the experimental results obtained.

The rest of this article is structured as follows. Section 2 reviews previous research on crowding in GAs and focuses on generalized crowding. Section 3 discusses several control methods for generalized crowding's scaling factor parameter, including the novel diversity-adaptive and self-adaptive methods. Section 4 presents experimental results for traditional, diversity-adaptive, and self-adaptive generalized crowding applied to real function optimization, Bayesian networks inference, and the Traveling Salesman Problem. These experimental results are discussed in detail in Section 5. Finally, Section 6 contains the main conclusions from this work and outlines future research.

2. Crowding techniques

In general, crowding techniques give rise to overlapping-generation evolutionary algorithms (parents and offspring compete for survival) with a replacement-oriented survival mechanism.

The original crowding scheme developed by de Jong [9] consists of randomly selecting, for each offspring, γ individuals from the current population. The offspring will replace the most similar selected individual. The parameter γ is known as the crowding factor, and usually $\gamma = 2$ is used.

de Jong's scheme was changed slightly by Mahfoud [25,26]. Since an offspring is likely to be similar to its parents, the following crowding GA scheme can be used to efficiently preserve diversity in the population:

1. The individuals in the current population are randomly paired. (Since parent selection is not usually applied under crowding, every individual in the population becomes a parent.)
2. With probability P_C (crossover probability), the parents in each pair (p_1, p_2) are recombined. The two resulting children (c_1, c_2) are mutated with probability P_M (mutation probability).
3. Each child competes with one of its two parents to be included in the population of the next generation. Let $d(i_1, i_2)$ denote the distance between two individuals, i_1 and i_2:

   If $d(p_1, c_1) + d(p_2, c_2) < d(p_1, c_2) + d(p_2, c_1)$

   $p_1 \leftarrow$ winner of competition between p_1 and c_1,

   $p_2 \leftarrow$ winner of competition between p_2 and c_2.

   else

   $p_1 \leftarrow$winner of competition between p_1 and c_2,

$p_2 \leftarrow$winner of competition between p_2 and c_1.

Under this scheme, each offspring competes for survival with its most similar parent, although other variants exist that match more than two parents and two children before the similarity metric is applied [30, Section 4.2]. The "family competition" idea has been the basis of a number of widely-used crowding algorithms [25,26,29,28,27], although other approaches like restricted tournament selection achieve a similar effect [18]. One key difference between these approaches is the rule used to decide the winner for each competition.

In crowding, the way a competition takes place between parent p and child c is defined through a so-called replacement rule. We now discuss two of the most widely used replacement rules: deterministic replacement and probabilistic replacement.

In Mahfoud's deterministic crowding [25,26], the winner of a competition between parent p and child c is the one with higher fitness. Let P_c denote the probability that child c replaces parent p in the population. This probability can be expressed in the following way for deterministic replacement:

$$P_c = \begin{cases} 1 & \text{if } f(c) > f(p) \\ 0.5 & \text{if } f(c) = f(p) \\ 0 & \text{if } f(c) < f(p) \end{cases},$$

where we assume maximization of a real-valued fitness function f. (Without loss of generality, maximization problems will be assumed throughout this article unless otherwise stated.)

Unlike deterministic crowding, probabilistic crowding as introduced by Mengshoel and Goldberg [29,28] uses a non-deterministic rule to establish the winner of a competition between parent p and child c. The probability that c replaces p in the population is the following:

$$P_c = \frac{f(c)}{f(c) + f(p)}.$$

Similar to the deterministic and probabilistic crowding schemes, *generalized crowding* introduced by Galan and Mengshoel [14] consists of a pairing phase and a replacement phase. The novelty is that the replacement phase relies on a *scaling factor* ϕ, which enables a broad range of replacement rules just by varying ϕ. In generalized crowding, the winner of a competition between parent p and child c is established using this formula:

$$P_c = \begin{cases} \frac{f(c)}{f(c)+\phi \cdot f(p)} & \text{if } f(c) > f(p) \\ 0.5 & \text{if } f(c) = f(p) \\ \frac{\phi \cdot f(c)}{\phi f(c)+f(p)} & \text{if } f(c) < f(p) \end{cases}, \tag{1}$$

where P_c is the probability that child c replaces parent p in the population, f is the fitness function to be maximized, and $\phi \in \Re^+ \cup \{0\}$ denotes the parameter named scaling factor.

The key idea in generalized crowding is that fitness scaling of the least fit individual, among p and c, is done before probabilistic crowding is applied. As shown in Eq. (1), if $f(p) < f(c)$ then $f(p)$ is transformed into $\phi \cdot f(p)$; otherwise, if $f(c) < f(p)$ then $f(c)$ is transformed into $\phi \cdot f(c)$.

When $\phi = 0$ in Eq. (1), generalized crowding becomes equivalent to deterministic crowding. When $\phi \in (0,1)$, it is possible that the least fit among p and c wins in the replacement phase. When $\phi = 1$, generalized crowding turns into probabilistic crowding. Finally, when $\phi \in (1, +\infty)$ the probability that the least fit of p and c wins is greater than in probabilistic crowding, which perhaps sounds counter-intuitive but can be beneficial for deceptive or highly multimodal problems [14].

To avoid having to deal with three separate cases in Eq. (1), one can use instead the logistic function $\ell(x) = 1/(1 + e^{-x})$ to obtain:

$$P_c = \frac{\phi^{\ell(f(p)-f(c))} \cdot f(c)}{\phi^{\ell(f(p)-f(c))} \cdot f(c) + \phi^{\ell(f(c)-f(p))} \cdot f(p)}.$$

The scaling factor ϕ allows a wide range of selective pressures to be applied: The greater ϕ is, the more probable it is that locally non-optimal parts of the search space are explored.

3. Scaling factor control

A GA's parameters can be either manually tuned in advance of its run or automatically controlled during execution. Compared to manual parameter tuning, advantages of automatic parameter control are that (i) it is less taxing on the GA's user and (ii) parameters can be adapted to the state of the search process, thus potentially producing better results. A classification of parameter setting techniques for evolutionary algorithms can be found in the book of Eiben and Smith [11, Chapter 8].

Setting the proper value of ϕ in generalized crowding GAs is crucial for good performance. At the same time, choosing the right ϕ-value is a hard and problem-dependent task [14, Section 5]. This section introduces two approaches for automatically controlling the ϕ parameter of generalized crowding: diversity-adaptive control and self-adaptive control. One main objec-

tive of this work is to study whether these approaches can improve the traditional method of setting ϕ prior to GA execution and keeping it fixed throughout the search process.

As a motivation for the introduction of diversity-adaptive and self-adaptive control of ϕ, we first describe nonadaptive control, a simple but limited method for varying ϕ. Nonadaptive control of the scaling factor ϕ consists of changing ϕ by means of a deterministic rule. This rule is predetermined by the user, remains fixed throughout the GA's execution, and uses no feedback from the GA population. Given that it is reasonable to gradually reduce the degree of exploration during a GA's run, a few examples of this control technique are the following:

- *Exponentially decaying scaling factor*. Under this scheme, the scaling factor at generation t, $\phi(t)$, is calculated by multiplying $\phi(t-1)$ by a decay constant $k \in [0, 1]$. In this way,

$$\phi(t) = \phi(1) \cdot k^{t-1},$$

where $\phi(1)$ is the scaling factor defined by the user for $t = 1$, the first generation of the GA. When $k = 1$ this scheme is identical to the traditional generalized crowding GA that keeps ϕ fixed [14]. Additionally, when $k = 0$ this scheme becomes equivalent to deterministic crowding. In general, the greater k is, the more exploration is applied.

- *Linearly decaying scaling factor*. Under this scheme, the scaling factor at generation t, $\phi(t)$, is calculated by subtracting from $\phi(t-1)$ a decay constant $k \in \Re^+ \cup \{0\}$. Therefore,

$$\phi(t) = \phi(1) - \{k \cdot (t-1)\},$$

where $\phi(1)$ is the scaling factor defined by the user for the first generation. Obviously, if the formula produces negative values then $\phi(t)$ is considered to be equal to zero. In this scheme, the greater k is, the less exploration is applied.

A disadvantage of these methods is that the user needs to assign a value to k before running the GA. Since k determines the degree of exploration applied, a similar problem as in the case of fixed ϕ arises: It is impossible to know in advance the proper degree of exploration for a given optimization problem, which forces the user to either pick a "reasonable" value for k or spend time to manually tune its value. This disadvantage appears also in crowding GAs applying a replacement rule based on simulated annealing [27,30], where deciding the proper cooling schedule for the temperature is a hard task. For these reasons, this work focuses on diversity-adaptive and self-adaptive control methods for ϕ, which require no additional parameters like k to be defined and, consequently, appear easier to use compared to nonadaptive methods.

Fig. 1 summarizes how the adaptive and nonadaptive control methods work. The traditional method applies a fixed scaling factor to the crowding process (see Fig. 1a); more generally, one of the exponentially or linearly decaying scaling factors discussed above can be used. The diversity-adaptive control method, discussed in Section 3.1, updates the scaling factor, prior to the crowding process, by using information on the diversity of the current population (see Fig. 1b). Finally, the self-adaptive control method, discussed in Section 3.2, incorporates the scaling factor in the chromosome of the individuals, and the scaling factors are independently subjected to initialization, recombination, and mutation (see Fig. 1c).

3.1. Diversity-adaptive control

Diversity-adaptive control of the scaling factor ϕ uses feedback from the GA population to determine the magnitude of the change to ϕ. The updating mechanism used to control ϕ is applied between consecutive GA generations. Specifically, ϕ can be set proportionally to the population diversity at each generation in the following way:

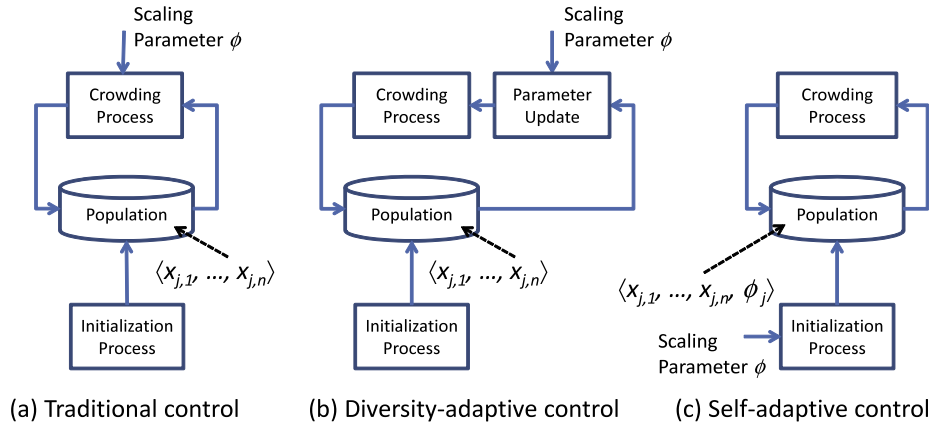

Fig. 1. Overview of how the traditional (a), diversity-adaptive (b), and self-adaptive (c) control methods work in a generalized crowding GA.

$$\phi(t) = \phi(1) \cdot \frac{\Delta(t)}{\Delta(1)}, \tag{2}$$

where $\phi(1)$ is the scaling factor defined by the user for the first generation, and $\Delta(t)$ is the population diversity at generation t. In this way, ϕ is lowered at the same rate as the population diversity decreases. Consequently, the degree of exploration is reduced as the GA progresses. The high values of $\phi(t)$ in the first generations prevent premature convergence to local optima with poor fitness. The low values of $\phi(t)$ in the last generations, when population diversity is nearly lost, favor that regions close to (local or global) optima are effectively exploited.

In this work, we use a diversity measure based on the concept of population entropy. Entropy has been used in the context of GAs to prevent premature convergence. For example, Farhang-Mehr and Azarm [12] introduce an entropy-based multi-objective genetic algorithm. Tsujimura and Gen [42] calculate the entropy of each gene for the individuals in the population and compare it with a threshold value; when the number of genes whose entropy is lower than the threshold is greater than a given amount, the diversity of the population is increased by a process of allele exchange. Finally, Liu et al. [24] calculate entropy from the fitness distribution in the population and the entropy value is employed to change the mutation probability: When the entropy is greater than 0.5, the mutation probability is decreased to facilitate exploitation. If entropy is smaller than 0.5, the mutation probability is increased to diversify the search process.

We use entropy as a measure to directly determine the value of the scaling factor, generation by generation. Specifically, we define $\Delta(t) = H(t)$ in Eq. (2), where $H(t)$ is the mean entropy (over the genes) for the individuals in the population at generation t. If the genotype is formed by n genes, the mean entropy at t is defined as follows:

$$H(t) = \frac{1}{n}\sum_{i=1}^{n} H_i(t),$$

where $H_i(t)$ is the entropy of gene i. $H_i(t)$ can be calculated in the following way:

$$H_i(t) = -\sum_{j=1}^{v_i} P_{ij} \cdot \log_{v_i} P_{ij}, \tag{3}$$

where v_i is the number of alleles (or possibles values) for gene i, and P_{ij} is the frequency of individuals in the population whose gene i takes the jth possible value for that gene. Regardless of the magnitude of v_i in Eq. (3), $H_i(t)$ always belongs to the interval [0, 1], and so does $H(t)$.

3.2. Self-adaptive control

Self-adaptive parameter control has been extensively performed by evolutionary algorithms. In evolution strategies [36,37], the mutation step sizes are self-adapted throughout the search process. Both mutation rate [3] and crossover [31] have been self-adapted as well. In mating for GAs, the self-adapted parameter is the mating index, which defines the particular mating preference of an individual during a GA's mating phase [15].

Self-adaptive control of the scaling factor ϕ can be carried out by encoding this parameter into the chromosome, and performing recombination and mutation on the ϕ-values in the population. In this way, the ϕ-values leading to better individuals will have a greater chance of surviving.

If an individual j is represented as $\langle x_{j,1}, \ldots, x_{j,n}\rangle$, its extended representation under the self-adaptive scaling factor scheme is $\langle x_{j,1}, \ldots, x_{j,n}, x_{j,n+1}\rangle$, where $x_{j,n+1} = \phi_j$ is the scaling factor. In other words, ϕ is now a local parameter, and each individual has an independent behavior in the replacement phase.

In self-adaptive generalized crowding, the replacement rule for parent $p \equiv \langle x_{p,1}, \ldots, x_{p,n}, \phi_p\rangle$ and child $c \equiv \langle x_{c,1}, \ldots, x_{c,n}, \phi_c\rangle$ operates so that the probability P_c that child c replaces parent p is:

$$P_c = \begin{cases} \frac{f(c)}{f(c)+\phi_p f(p)} & \text{if } f(c) > f(p) \\ 0.5 & \text{if } f(c) = f(p) \\ \frac{\phi_c f(c)}{\phi_c f(c)+f(p)} & \text{if } f(c) < f(p) \end{cases}.$$

Note that in self-adaptive generalized crowding, the scaling factor applied in the replacement rule is that of the least fit individual among p and c. This is in accordance with generalized crowding, where the fitness of the worse individual among p and c is the one multiplied by the scaling factor.

It remains to consider how scaling factors are initialized, recombined, and mutated. As far as initialization is concerned, each scaling factor is assigned a real number generated uniformly at random from the range $[0, \phi_{\max}]$, where $\phi_{\max}$ is defined by the user. Recombination of the scaling factors of two parents can be carried out in several ways: by assigning to the children the mean of the parents' scaling factors, or by letting the children inherit the parents' scaling factors, among other possibilities. This work uses the latter method, since we have found that it produces better experimental results. Mutation of scaling factor is implemented by adding to its real value an amount drawn randomly from a Gaussian distribution with mean zero and standard deviation $\phi_{\max} \cdot 0.1$. In the case that the mutated scaling factor is located outside the interval $[0, \phi_{\max}]$, the original scaling factor is kept unchanged.

4. Experiments

The main purpose of this experimental section is twofold: on the one hand, we investigate whether generalized crowding with fixed scaling factor is outperformed by diversity-adaptive and self-adaptive generalized crowding; on the other hand, we conduct a comparative experimental study between diversity-adaptive and self-adaptive generalized crowding.

In this section, we compare the performance of these three scaling-factor control techniques in the context of generalized crowding:

1. Fixed scaling factor, as in traditional generalized crowding [14].
2. Diversity-adaptive scaling factor based on population entropy (see Section 3.1).
3. Self-adaptive scaling factor (see Section 3.1).

For each control technique, the user needs to set the value of a parameter before the GA is started. For simplicity, we call this parameter ϕ_0, and its meaning is different depending on the control technique. When fixed scaling factor is used, as in traditional generalized crowding, ϕ_0 is the ϕ-value that is kept constant for all GA generations ($\phi(t) = \phi_0$ for $t \geqslant 1$). In the case of diversity-adaptive scaling, ϕ_0 is the initial scaling factor at the first generation ($\phi(1) = \phi_0$). If self-adaptive scaling is employed, then ϕ_0 is the maximum scaling factor that can be assigned to individuals ($\phi_{\max} = \phi_0$).

When designing a GA, one has to decide what variation and selection methods to use among a set of possibilities; furthermore, each method may include parameters whose values have to be set by the user. In our case, we have made these choices based on the literature on GAs, our accumulated knowledge, and preliminary experiments carried out as part of this work. Obviously, finding the optimal configuration of methods and parameters values is problem-dependent and a highly complex task, and our claim is only to have employed approximately optimal configurations rather than optimal configurations in the experiments.

For the sake of generality, several distinct domains leading to different genotype representations were experimented with: (1) real-function optimization for which we use a GA with real-valued representation; (2) Bayesian network inference for which we employ a GA with integer encoding of genes values; and (3) the Traveling Salesman Problem where individuals are represented through permutations of different elements.

4.1. Real function optimization

Given an n-dimensional function, $f : \Re^n \to \Re$, global optimization consists of determining $x^* \in \Re^n$ such that $f(x^*) \geqslant f(x)$ for all $x \neq x^*$ with $x \in \Re^n$. This definition corresponds to a maximization problem. In the case of minimization, the inequality is $f(x^*) \leqslant f(x)$.

The experiments in this section were performed by means of a GA using n real-coded variables as genotype, no parent selection, random mating, uniform crossover [40] with crossover probability equal to one for each pair of parents, uniform mutation (that draws a value uniformly at random from the interval where the corresponding real variable is defined) with a given low mutation probability for each gene or real variable, generational survivor selection, and no kind of elitism. Different seeds for the random number generator were used for each run of the GA.

In preliminary experiments with generalized crowding using a fixed scaling factor, we extensively tested various unimodal and multimodal fitness functions. We identified the following two groups of functions:

- *Group 1*: Functions for which the lower ϕ is, the better is the performance obtained. This means that a more exploitative generalized crowding, or lower ϕ, tends to produce better results for this kind of function. Typical multimodal functions like Michalewicz [32], Schwefel [37], Rastrigin [33], and Ackley [2] turned out to belong to this group. We obtained this result from several experiments (partly reported at the end of this section) for different dimensions, population sizes, and crossover rates, among other parameters.
- *Group 2*: Functions for which the best performance is obtained for some specific $\phi > 0$, and not for $\phi = 0$ as in the other group. Finding functions belonging to this group was more difficult in comparison to group 1; below we enumerate several of them.

For the sake of generality, we experimented extensively with functions from each group. Previously, six two-dimensional functions F_1–F_6 were used by Ballester and Carter [4] to empirically demonstrate that GAs with random selection of parents and crowding replacement are robust optimizers, whereas GAs with tournament selection of parents and random replacement perform poorly in comparison. The present work constitutes a further step in this direction, since new, adaptive, and more consistent crowding GAs are introduced. Our preliminary experiments showed that Ballester and Carter's F_1, F_3, and F_6 (see [4, Section 4]) belong to Group 1, whereas F_2, F_4, and F_5 belong to Group 2. We chose for our extensive experiments presented below the functions F_1 and F_2, which are defined over the range $-10 \leqslant x_1, x_2 \leqslant 10$, as follows:

$$F_1(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$$
$$F_2(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3.$$

Since these two functions were designed for minimization, we transformed them (without loss of generality) so that maximization is applied and their values are always positive over range $[-10, 10]$, as required by Eq. (1). The resulting transformed functions used in this work are the following:

$$F_1^*(x_1, x_2) = -F_1(x_1, x_2) + 301.4$$
$$F_2^*(x_1, x_2) = -F_2(x_1, x_2) + 300.6.$$

Since $x_1^2 + 2x_2^2 \leqslant 300$ for $-10 \leqslant x_1, x_2 \leqslant 10$ and $-1 \leqslant \cos(\cdot) \leqslant 1$, the election of constants 301.4 and 300.6 guarantees that F_1^* and F_2^* are positive. In this way, the optimal value for F_1^* is 301.4, and the optimal value for F_2^* is 300.6. For simplicity, in the rest of the article we will refer to F_1^* and F_2^* as F_1 and F_2 respectively.

Table 1 summarizes the GA configuration in the experiments for F_1 and F_2. Fig. 2 represents the evolution, generation by generation, of the mean best fitness (considering the best individual found from the first generation to the current generation) for function F_1. Three cases are depicted in Fig. 2: fixed scaling (Fig. 2a and d), diversity-adaptive scaling (Fig. 2b and e), and self-adaptive scaling (Fig. 2c and f). Fig. 3 depicts, for F_2, the same metrics as Fig. 2 depicts for F_1. Note that the case of fixed scaling factor for $\phi_0 = 0$ corresponds to deterministic crowding, and the case of fixed scaling factor for $\phi_0 = 1$ corresponds to probabilistic crowding.

We next consider the Michalewicz and Schwefel functions. The Michalewicz function [32] is defined as follows:

$$f_{\text{Michalewicz}}(x) = \sum_{i=1}^{n} \sin(x_i) \cdot \left\{ \sin\left(\frac{i \cdot x_i^2}{\pi} \right) \right\}^{2 \cdot m} \text{ with } 0 \leqslant x_i \leqslant \pi,$$

where m usually takes on a value of 10. The Schwefel function [37] is defined as:

$$f_{\text{Schwefel}}(x) = \sum_{i=1}^{n} x_i \cdot \sin\left(\sqrt{|x_i|} \right) \text{with } -500 \leqslant x_i \leqslant 500$$

and has a maximum at $(x_1 = 420.9687, \ldots, x_n = 420.9687)$ whose value is $n \cdot 418.9829$. We added $500 \cdot n$ to the Schwefel function in order to transform it into a positive function.

Fig. 4 shows experimental results for the Michalewicz and the Schwefel functions. The experiments in Fig. 4 were carried out for $n = 80$ and $M = 100$, and produced results qualitatively similar to those of F_1 in Fig. 2.

4.2. Bayesian network optimization

A Bayesian network (BN) [34,20,7] is an acyclic directed graph whose nodes represent random variables and whose arcs establish probabilistic dependence and independence relations between the random variables. In this article, we only consider discrete random variables. The parametric part of a BN is formed by a set of conditional probability tables (CPTs) that contain the probabilities of a node's states given any possible configuration of values for its parents. For root nodes, only prior probabilities are needed. Given a BN with nodes $V = \{V_1, \ldots, V_n\}$, the joint probability over the random variables in the network is defined as follows:

$$P(v_1, \ldots, v_n) = \prod_{i=1}^{n} P(v_i | pa(v_i)),$$

where $pa(v_i)$ stands for a configuration of the set of parents for variable V_i. Note that a random variable is represented by an uppercase letter, whereas a value (or state) it can take is denoted by the same lowercase letter.

This section deals with abductive inference in BNs, also known as belief revision. Given some evidence or set of observations, $\mathbf{e} = \{E_1 = e_1, \ldots, E_m = e_m\}$ with $E_i \in \{V_1, \ldots, V_n\}$ $\forall i$, an explanation is defined as a configuration of values for the network variables, $\mathbf{v} = \{V_1 = v_1, \ldots, V_n = v_n\}$, such that $\mathbf{e} \subset \mathbf{v}$. The goal of abductive inference is to find an explanation $\mathbf{v}^*$ with maximum posterior probability:

Table 1
GA configuration for the experiments in Figs. 2 and 3.

GA parameters	Values
Population size	$M = 20$
Genes per individual	$n = 2$
Mating	Random
Crossover	uniform ($P_C = 1$)
Mutation	uniform ($P_M \in \{0.0125, 0.025\}$)
Survivor selection	Generational
Scaling parameter	$\phi_0 \in \{0, 0.25, 0.5, 0.75, 1, 1.25\}$
Entropy intervals	100 per dimension
Runs number	1000

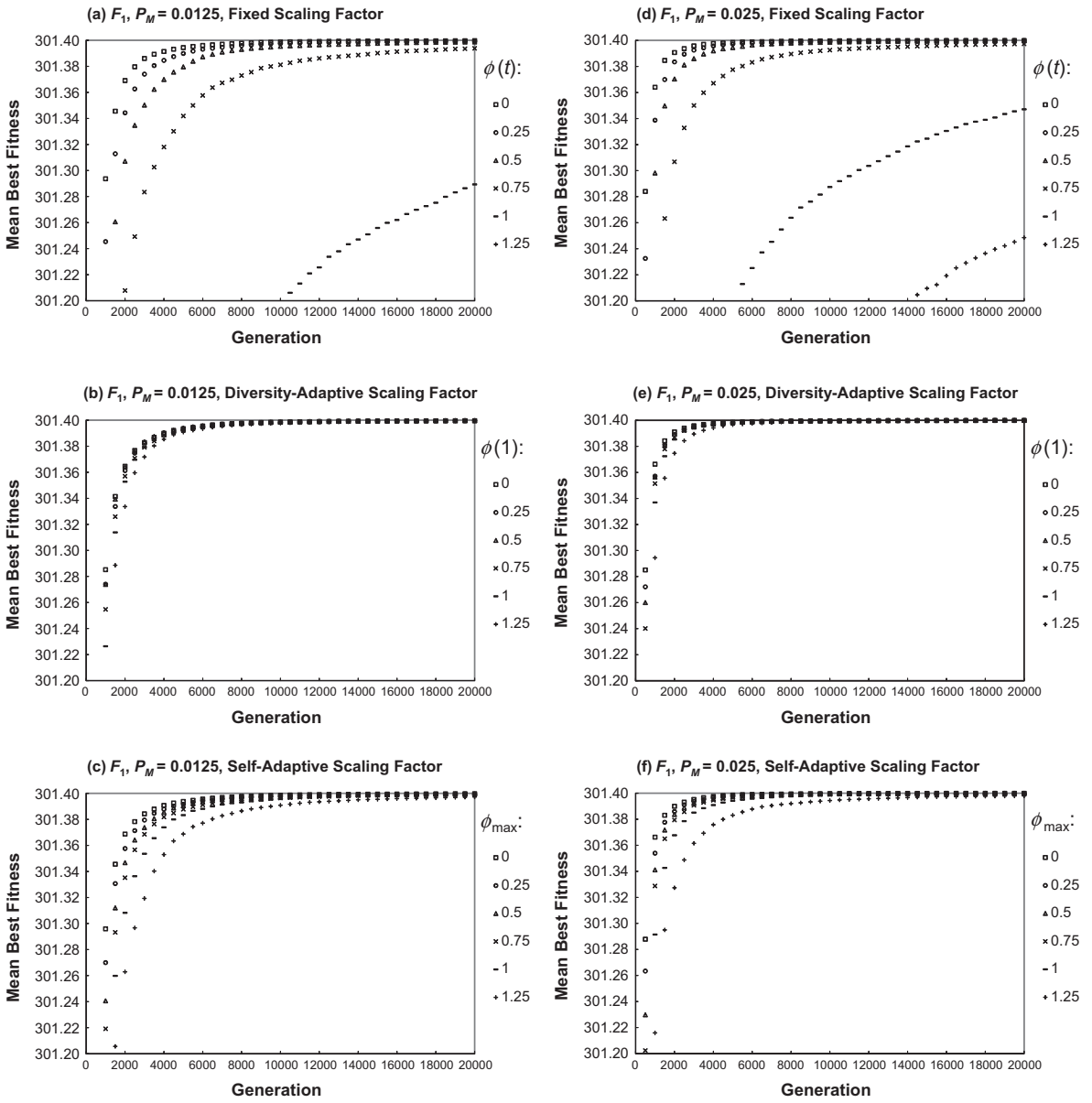

Fig. 2. Mean best fitness results over 1000 runs for F_1 with population size $M = 20$ and mutation probabilities: $P_M = 0.0125$ (left graphs) and $P_M = 0.025$ (right graphs). Cases studied: fixed scaling factor (graphs (a) and (d)), diversity-adaptive scaling factor (graphs (b) and (e)), and self-adaptive scaling factor (graphs (c) and (f)). Each graph contains tests for $\phi_0 \in \{0, 0.25, 0.5, 0.75, 1, 1.25\}$.

$$\mathbf{v}^* = \arg \max_{\mathbf{v}} P(\mathbf{v}|\mathbf{e}),$$

which is usually referred to as a *most probable explanation*.

The estimation of a most probable explanation in BNs is of interest in many applications including diagnosis, image recognition, genetic linkage analysis, and error correction decoding. Abductive inference in BNs has been proven to be an $\mathcal{NP}$-hard problem [38], but some approximate and heuristic methods exist that allow optimal or approximately optimal explanations to be computed even for complex networks. The present section deals with approximate methods that use evolutionary computing to perform abductive inference in BNs [13]. We employ a GA in which the representation of individuals (or explanations) uses a string whose elements are defined over a finite alphabet, such that each position (or gene) in the string corresponds to one variable in the BN. Each gene can take on a number of discrete values (or alleles) which correspond to the domain values of its associated random variable.

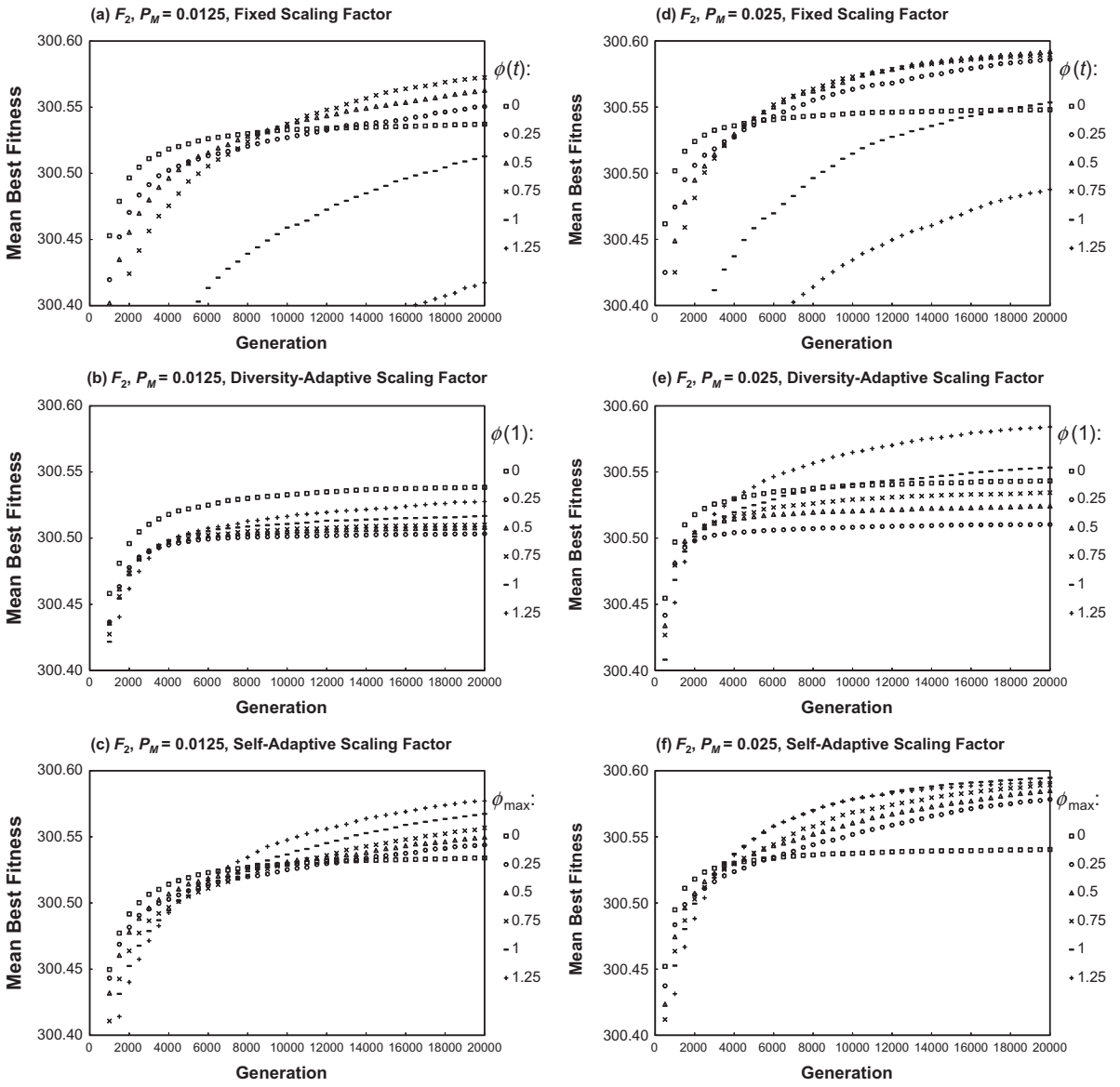

Fig. 3. Mean best fitness results over 1000 runs for F_2 with population size $M = 20$ and mutation probabilities: $P_M = 0.0125$ (left graphs) and $P_M = 0.025$ (right graphs). Cases studied: fixed scaling factor (graphs (a) and (d)), diversity-adaptive scaling factor (graphs (b) and (e)), and self-adaptive scaling factor (graphs (c) and (f)). Each graph contains tests for $\phi_0 \in \{0, 0.25, 0.5, 0.75, 1, 1.25\}$.

We used in our experiments variants of the well-known Alarm BN [5], which has 37 nodes and 752 conditional probabilities. As done previously [13], we randomly introduced zeros into the CPTs of the Alarm BN in order to vary its search space characteristics. Specifically, zeros were introduced with probability $\rho \in \{0.0, 0.2\}$ as reflected in the notation Alarmρ. An Alarmρ BN is not the same as the Alarm BN; they have the same graph structure, node cardinalities, and number of CPT values, but CPT values are generated in a randomized way for Alarmρ, where the number of CPT values equal to zero is approximately $\rho \cdot 752$.

We varied ϕ_0 in our experiments with Alarmρ BNs and investigated $\phi_0 \in \{0, 10, 20, 30, 40\}$. Table 2 contains the GA configuration employed in these experiments. Uniform crossover was used with crossover probability $P_C = 1$. For a BN with n nodes, we used mutation probability for each gene given by $P_M = 1/n$, and mutation amounted to changing a node's state uniformly at random. Experimental results were averaged over one thousand runs. The results of the experiments are summarized in Fig. 5.

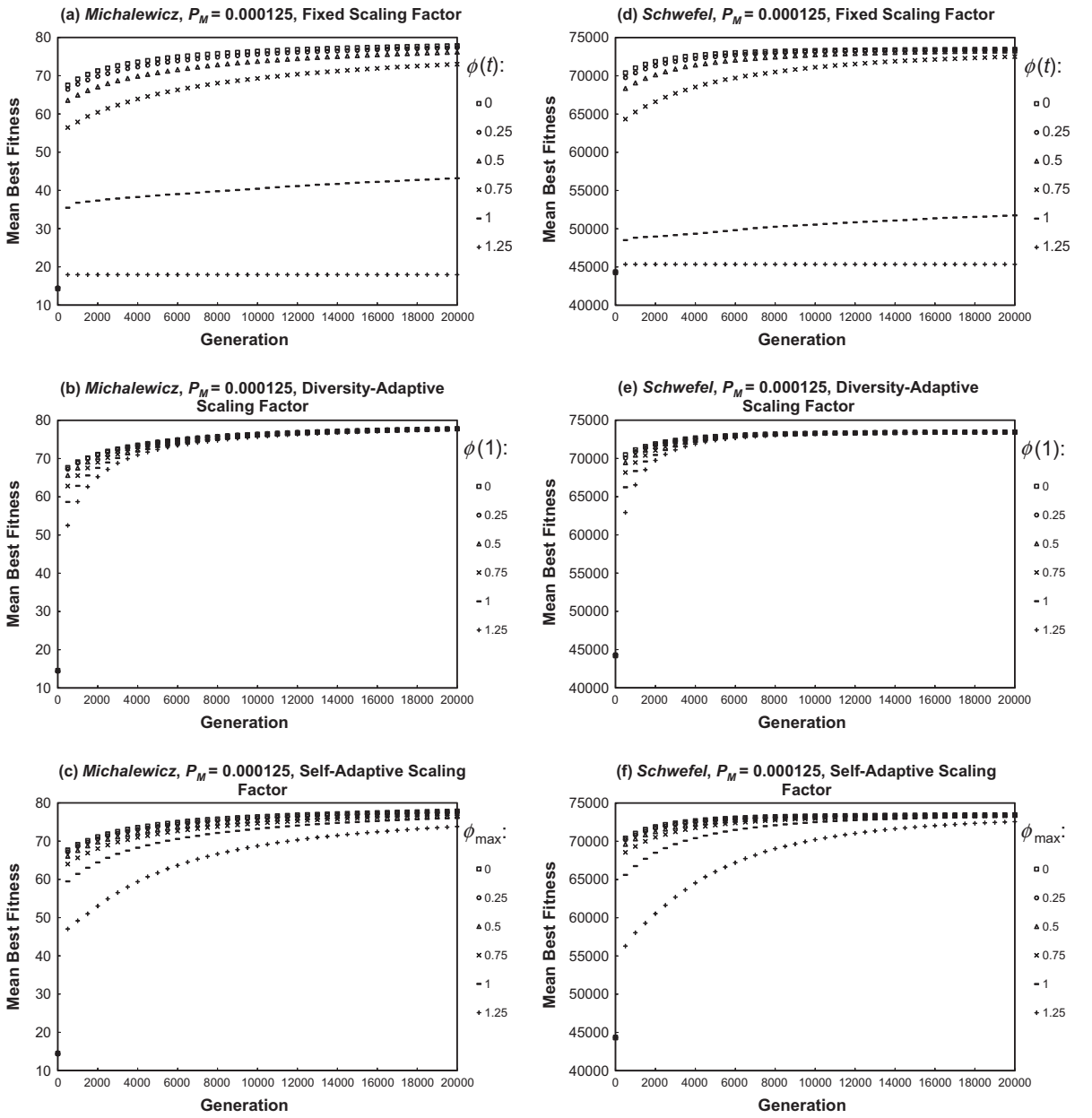

Fig. 4. Mean best fitness results over 100 runs for the Michalewicz (left graphs) and Schwefel (right graphs) functions with dimension $n = 80$, population size $M = 100$, and mutation probability $P_M = 0.000125$. Cases studied: fixed scaling factor (graphs (a) and (d)), diversity-adaptive scaling factor (graphs (b) and (e)), and self-adaptive scaling factor (graphs (c) and (f)). Each graph contains tests for $\phi_0 \in \{0, 0.25, 0.5, 0.75, 1, 1.25\}$.

Table 2
GA configuration for the experiments in Fig. 5.

GA parameters	Values
Population size	$M = 20$
Genes per individual	$n = 37$
Mating	Random
Crossover	uniform ($P_C = 1$)
Mutation	random ($P_M = \frac{1}{n} = \frac{1}{37}$)
Survivor selection	Generational
Scaling parameter	$\phi_0 \in \{0, 10, 20, 30, 40\}$
Runs number	1000

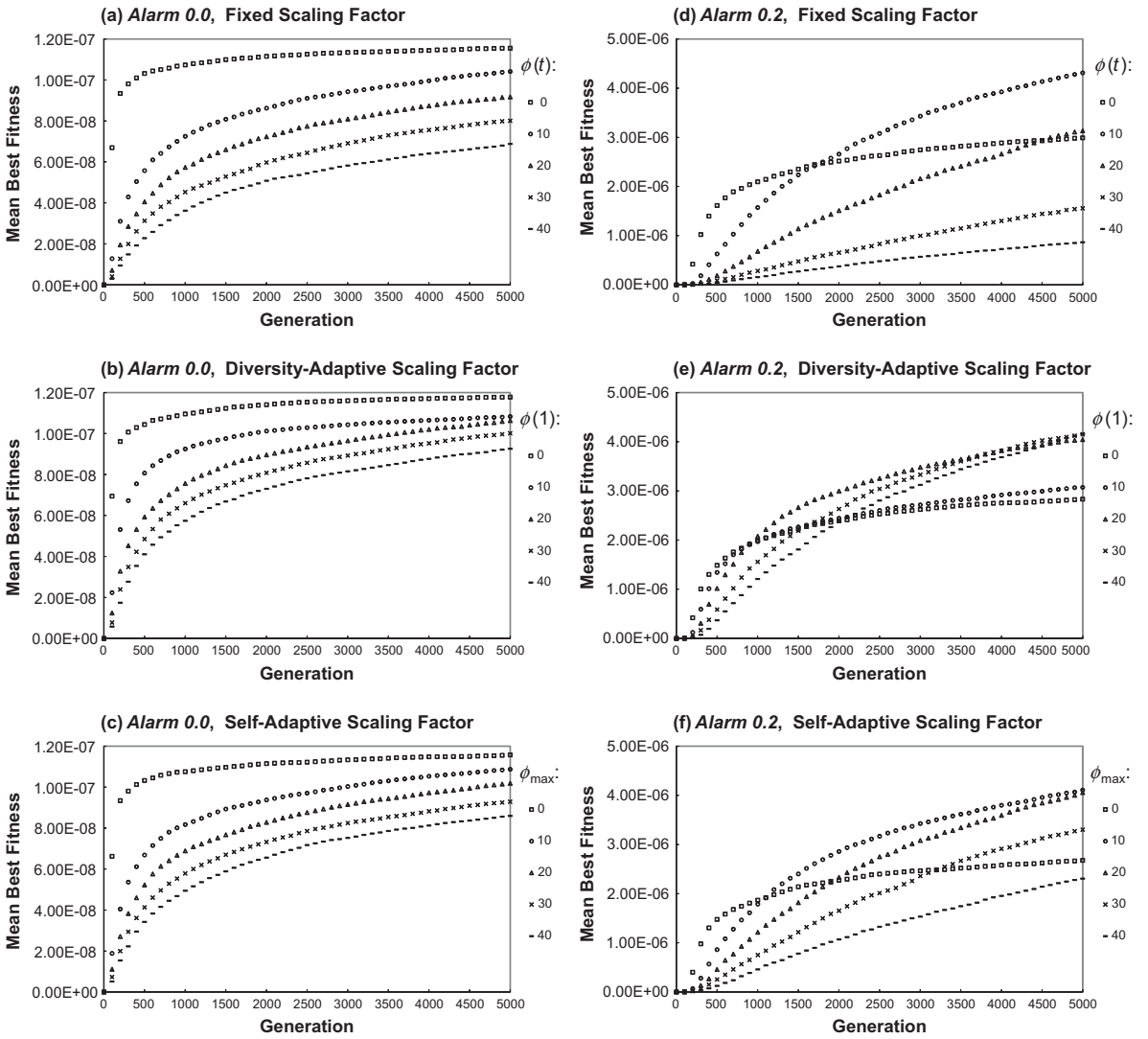

Fig. 5. Mean best fitness results over 1000 runs for Alarm0.0 (left graphs) and Alarm0.2 (right graphs) with population size $M = 20$. Cases studied: fixed scaling factor (graphs (a) and (d)), diversity-adaptive scaling factor (graphs (b) and (e)), and self-adaptive scaling factor (graphs (c) and (f)). Each graph contains tests for $\phi_0 \in \{0, 10, 20, 30, 40\}$.

4.3. Traveling salesman problem

The Traveling Salesman Problem (TSP) is a combinatorial optimization problem which has been widely studied in the scientific literature on evolutionary computation [17,8,43,41,23]. The TSP consists of finding a path of minimum length between several cities, such that any possible path visits each of the cities only once, departing and arriving at the same city.

We employed in the experiments two problems of different complexity. The first problem, called TSPsquare16, is a simple TSP consisting of 16 cities placed as a square, such that there are five cities on each side of the square. Since each city is separated one hundred units from its nearest city, the optimum has a value of 1600. The second problem, called TSPBerlin52, is a more complex TSP consisting of 52 locations in Berlin.[1] In this case, the optimum path has a length of 7542. These two problems are different in one important aspect: While our GAs reach the optimum for TSPsquare16 in some of the experiments, they suffer from premature convergence in all the experiments for TSPBerlin52. We are thus able to study two main classes of behavior that can be found in this domain. Another important point is the fitness function, which should be adapted to maximization when crowding is applied; in order to do that, we internally used in our GAs the inverse of path length as fitness function. Nonetheless, for the sake of clarity, we still use path length in the graphs to illustrate the results obtained in the experiments.

[1] We obtained this problem from the TSP library that can be found at: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

Table 3
GA configuration for the experiments in Fig. 6.

GA parameters	Values
Population size	$M = 100$
Genes per individual	$n \in \{16, 52\}$
Mating	Random
Crossover	Modified order crossover ($P_C = 0.9$)
Mutation	Swap mutation ($P_M = 0.6$)
Survivor selection	Generational
Scaling parameter	$\phi_0 \in \{0, 0.02, 0.04, 0.06, 0.08, 0.1\}$
Runs number	1000

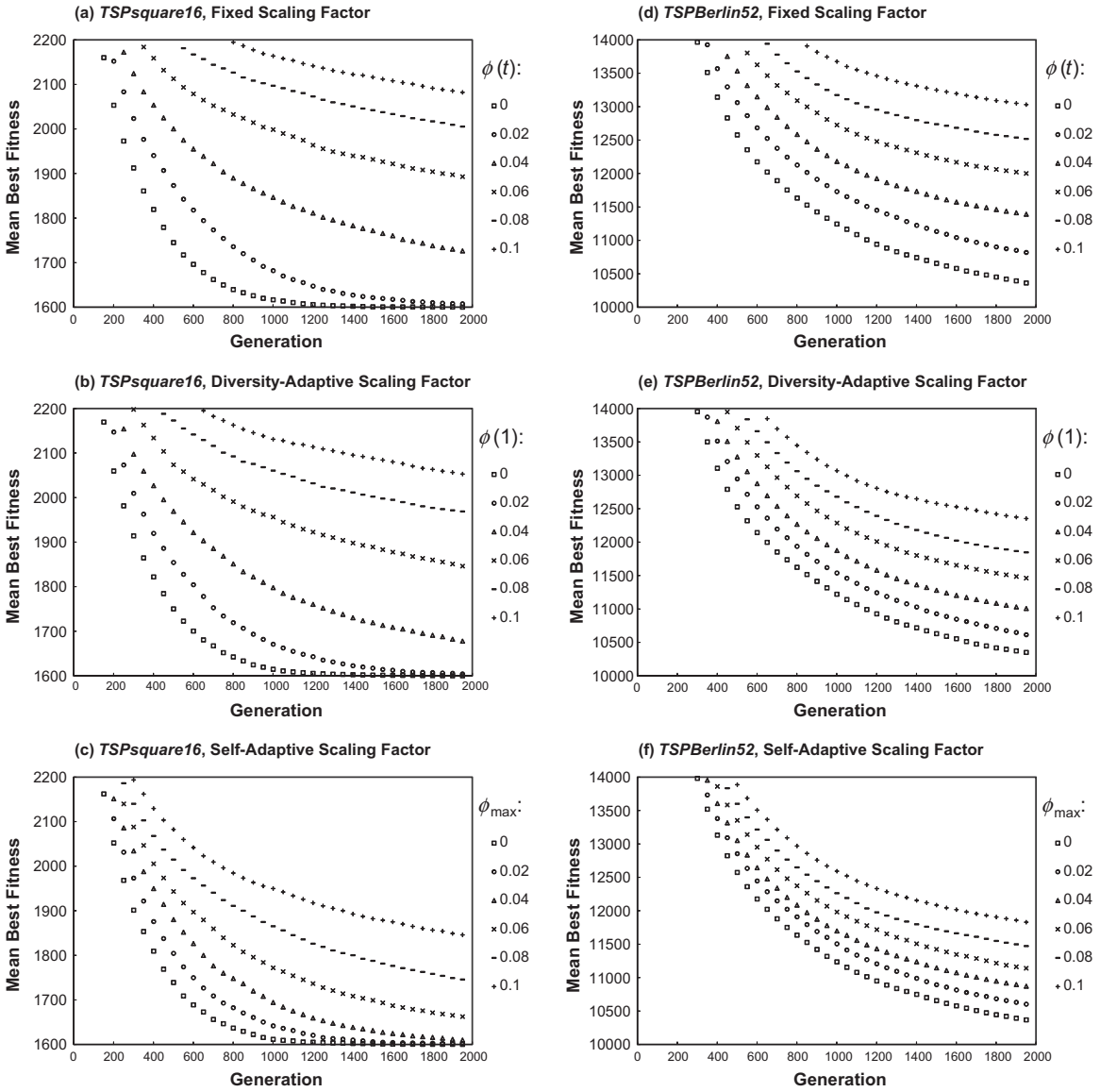

Fig. 6. Mean best fitness results over 1000 runs for TSPsquare16 and TSPBerlin52 with population size $M = 100$. Cases studied: fixed scaling factor (graphs (a) and (d)), diversity-adaptive scaling factor (graphs (b) and (e)), and self-adaptive scaling factor (graphs (c) and (f)). Each graph contains tests for $\phi_0 \in \{0, 0.02, 0.04, 0.06, 0.08, 0.1\}$.

Experiments were carried out with a GA using permutation of cities as genotype, no parent selection, random mating, modified order crossover [44] with crossover probability 0.9, swap mutation [41] with mutation probability for each individual of 0.6, and generational survivor selection. Table 3 shows the GA configuration for these experiments.

Fig. 6 depicts the results obtained for TSPsquare16 and TSPBerlin52. The population size was set to $M = 100$, one thousand runs were completed for each particular experiment, and no kind of elitism was used. As in our previous experiments, three cases are shown in Fig. 6: fixed scaling factor (Fig. 6a and d), diversity-adaptive scaling factor (Fig. 6[*b*] and [*e*]), and self-adaptive scaling factor (Fig. 6c and f). Each graph in Fig. 6 contains tests for $\phi_0 \in \{0, 0.02, 0.04, 0.06, 0.08, 0.1\}$.

5. Discussion of experiments

The experimental study carried out in Section 4 used different domains leading to different genotype representations and genetic operators. We also tried to cover problems of varying complexity in each domain and ϕ-values contained in distinct ranges; for example, in the real functions domain $\phi \in [0, 1.25]$, in the BNs domain $\phi \in [0, 40]$, and in the TSP domain $\phi \in [0, 0.1]$. For each domain and the case of fixed ϕ, we tried to cover, on the one hand, problems in which lower ϕ leads to better results and, on the other hand, problems where the best behavior is not obtained when $\phi = 0$. The experimental results obtained in Section 4 for the three domains studied in this article are summarized in Table 4. The table focuses on the mean best fitness produced in the final GA generation.

Before discussing the experimental results obtained in Section 4, their statistical significance needs to be established. In other words, for each figure in Section 4 (from Figs. 2a–6f), it has to be determined whether or not the graphs in the figure come from the same distribution and, therefore, the differences are due to random effects (null hypothesis). ANOVA tests were performed on the best-fitness data for the last generation depicted in each of the graphs contained in Figs. 2a–6f; for each graph, one thousand samples were taken. Table 5 contains the results for ANOVA tests on groups of samples constituted by mean best fitness values under $\phi_0 \in \{0, 0.25, 0.5, 0.75, 1, 1.25\}$ for real-function optimization (Figs. 2a–3f), under $\phi_0 \in \{0, 10, 20, 30, 40\}$ for BN inference (Fig. 5a–f), and under $\phi_0 \in \{0, 0.02, 0.04, 0.06, 0.08, 0.1\}$ for TSP (Fig. 6a–f). Besides the F value, the probability p of the corresponding result assuming the null hypothesis is shown. When probability p is less than a small threshold (typically, $p = 0.05$ is used in the literature), one is justified in rejecting the null hypothesis. In other words, the results are significant at the 5% significance level when $F > F_{\text{critical}}$. The values for p in Table 5 are small enough to discard the null hypothesis for all the experiments.

5.1. Real function optimization

Note that, for function F_1, both diversity-adaptive control of ϕ (Fig. 2b and e) and self-adaptive control of ϕ (Fig. 2c and f) outperform generalized crowding using fixed ϕ (Fig. 2a and c). The greater ϕ is, the more improvement is obtained. Furthermore, diversity-adaptive control of ϕ produces better results than self-adaptive control.

When F_2 is used as fitness function, on the one hand, generalized crowding using fixed ϕ (Fig. 3a and d) gives, compared to diversity-adaptive control of ϕ (Fig. 3b and e), better results for $\phi_0 \in \{0.25, 0.5, 0.75\}$, comparable results for $\phi_0 = 1$, and much worse results for $\phi_0 = 1.25$. On the other hand, generalized crowding using fixed ϕ (Fig. 3a and d) produces, compared to self-adaptive control of ϕ (Fig. 3c and f), comparable results for $\phi_0 \in \{0.25, 0.5, 0.75\}$ and much worse results for $\phi_0 \in \{1, 1.25\}$. Interestingly, a different result appears in the case of F_2 compared to F_1: Diversity-adaptive control of ϕ now produces worse results than self-adaptive control.

Overall, Figs. 2 and 3 suggest that adaptive control of ϕ is much more consistent to ϕ_0 changes than what fixed ϕ is. Furthermore, whereas fixed ϕ and adaptive ϕ are nearly comparable in the case of low ϕ-values, for the case of larger ϕ-values the performance of fixed ϕ greatly deteriorates compared to that of adaptive ϕ.

5.2. Bayesian network optimization

The experimental BN results suggest that, as the number of zeros in the CPTs grows and consequently the complexity of the search space increases (with more plateaus of fitness, or probability, equal to zero), it is useful to increase ϕ under fixed scaling factor in order to increase the degree of GA exploration. For example, in Fig. 5d, a fixed $\phi = 10$ yields much better results than a fixed $\phi = 0$. On the other hand, as in the case of real function optimization included in Section 4.1, both diversity-adaptive control of ϕ (Fig. 5b and e) and self-adaptive control of ϕ (Fig. 5c and f) produce better global optimization results than generalized crowding using fixed ϕ (Fig. 5a and d). Consistency is also higher for the adaptive techniques, since their graphs are closer to each other. Moreover, diversity-adaptive control of ϕ produces better results than self-adaptive control in Fig. 5; this constitutes the same behavior as that obtained for real function F_1 and the opposite behavior to that obtained for F_2.

5.3. Traveling salesman problem

The results obtained for TSP in Fig. 6 clearly show that adaptive control of ϕ (Fig. 6b, c, e, and f) significantly improves on fixed ϕ (Fig. 6a and b) in terms of performance and consistency; furthermore, this fact is independent of the type of TSP problem: one in which the optimum is reached (TSPsquare16 in Fig. 6) or one in which premature convergence takes place (TSPBerlin52 in Fig. 6).

Table 4
Mean best fitness results in the final GA generation for the experiments in Figs. 2–6. (The optimum mean value in each row appears in bold, and standard deviations are included in parentheses.)

Optimization problem	P_M	ϕ_0	Mean best fitness Scaling-factor control methods		
			Nonadaptive (Fixed)	Diversity-adaptive	Self-adaptive
F_1	0.0125	0	**301.399 (7.72E−4)**	**301.399 (5.29E−4)**	**301.399 (5.36E−4)**
		0.25	**301.399 (1.08E−3)**	**301.399 (6.45E−4)**	**301.399 (7.18E−4)**
		0.5	301.398 (2.13E−3)	**301.399 (6.65E−4)**	**301.399 (1.34E−3)**
		0.75	301.393 (8E−3)	**301.399 (7.52E−4)**	**301.399 (1.79E−3)**
		1	301.289 (0.111)	**301.399 (6.58E−4)**	301.398 (1.76E−3)
		1.25	301.106 (0.184)	**301.399 (7.75E−4)**	301.397 (3.99E−3)
	0.025	0	**301.399 (1.61E−4)**	**301.399 (1.67E−4)**	**301.399 (1.56E−4)**
		0.25	**301.399 (3.99E−4)**	**301.399 (1.56E−4)**	**301.399 (2.01E−4)**
		0.5	**301.399 (5.28E−4)**	**301.399 (1.54E−4)**	**301.399 (2.93E−4)**
		0.75	301.397 (3.53E−3)	**301.399 (1.86E−4)**	**301.399 (3.63E−4)**
		1	301.347 (0.059)	**301.399 (2.32E−4)**	**301.399 (6.38E−4)**
		1.25	301.248 (0.137)	**301.399 (3.37E−4)**	301.397 (2.86E−3)
F_2	0.0125	0	300.537 (0.096)	**300.538 (0.098)**	300.533 (0.097)
		0.25	**300.550 (0.08)**	300.503 (0.104)	300.543 (0.083)
		0.5	**300.562 (0.069)**	300.508 (0.104)	300.549 (0.081)
		0.75	**300.572 (0.036)**	300.510 (0.103)	300.556 (0.074)
		1	300.512 (0.074)	300.516 (0.099)	**300.567 (0.065)**
		1.25	300.417 (0.102)	300.527 (0.097)	**300.577 (0.038)**
	0.025	0	**300.547 (0.096)**	300.543 (0.091)	300.540 (0.093)
		0.25	**300.586 (0.043)**	300.510 (0.101)	300.578 (0.05)
		0.5	**300.591 (0.025)**	300.524 (0.099)	300.584 (0.043)
		0.75	**300.589 (0.013)**	300.534 (0.092)	**300.589 (0.027)**
		1	300.553 (0.047)	300.553 (0.077)	**300.594 (0.013)**
		1.25	300.487 (0.082)	300.584 (0.042)	**300.591 (0.012)**
Alarm0.0	$\frac{1}{37}$	0	1.15E−7 (3.61E−8)	**1.18E−7 (3.56E−8)**	1.16E−7 (3.78E−8)
		10	1.04E−7 (2.69E−8)	1.08E−7 (3.94E−8)	**1.09E−7 (3.29E−8)**
		20	9.17E−8 (2.15E−8)	**1.06E−7 (2.91E−8)**	1.02E−7 (2.37E−8)
		30	8.01E−8 (2.06E−8)	**1.00E−7 (2.43E−8)**	9.29E−8 (2.08E−8)
		40	6.87E−8 (2E−8)	**9.26E−8 (2.21E−8)**	8.59E−8 (2.13E−8)
Alarm0.2	$\frac{1}{37}$	0	**2.99E−6 (2.77E−6)**	2.83E−6 (2.54E−6)	2.68E−6 (2.63E−6)
		10	**4.31E−6 (2.73E−6)**	3.07E−6 (2.71E−6)	4.11E−6 (2.77E−6)
		20	3.13E−6 (2.12E−6)	4.04E−6 (2.79E−6)	**4.05E−6 (2.56E−6)**
		30	1.56E−6 (1.39E−6)	**4.15E−6 (2.79E−6)**	3.30E−6 (2.12E−6)
		40	8.65E−7 (7.62E−7)	**4.16E−6 (2.61E−6)**	2.31E−6 (1.78E−6)
TSPsquare16	0.6	0	**1600 (0)**	**1600 (0)**	**1600 (0)**
		0.02	1606.45 (35.34)	1603.48 (24.96)	**1601.31 (15.12)**
		0.04	1722.59 (122.86)	1674.56 (103.61)	**1607.88 (34.34)**
		0.06	1888.63 (141.03)	1842.9 (139.95)	**1659.68 (95.4)**
		0.08	2002.38 (137.09)	1966.8 (137.92)	**1741.81 (125.33)**
		0.1	2079.55 (130.75)	2049 (138.52)	**1842.77 (138.62)**
TSPBerlin52	0.6	0	10338.1 (546.74)	**10324.69 (542.96)**	10339.64 (550.69)
		0.02	10792.3 (554.34)	10588.62 (570.67)	**10579.98 (563.44)**
		0.04	11368.54 (555.8)	10977.97 (597.75)	**10842.24 (570.88)**
		0.06	11977.92 (552.43)	11428.98 (614.34)	**11114.52 (563.87)**
		0.08	12495.87 (559.26)	11822.98 (589)	**11448.94 (569.24)**
		0.1	13012.7 (541.4)	12332.29 (620.39)	**11811.83 (575.1)**

Fig. 6b, c, e, and f demonstrate that, for TSP, self-adaptive control of ϕ outperforms control of ϕ through population diversity. This behavior is similar to that observed for F_2 in the domain of real function optimization.

5.4. Scaling factor evolution

It is interesting to study how the ϕ-values change over generations depending on the optimization problem considered, the type of adaptive control applied to ϕ, and the values of ϕ_0. For the sake of brevity, we selected the BN optimization domain in order to illustrate this change. Fig. 7 depicts the (averaged over 1000 runs) current scaling factor in the population for the diversity-adaptive case and the (averaged over 1000 runs) mean scaling factor in the population for the self-adaptive case. In this way, Fig. 7a corresponds to Figs. 5, 7b corresponds to Figs. 5, 7c corresponds to Fig. 5e, and 7d corresponds to Fig. 5f.

Table 5
ANOVA test results for the experiments reported in Section 4. When $F \geqslant F_{\text{critical}}$, the result is significant at the level of $p = 0.05$.

Figure	Problem	F	F_{critical}	p
Fig. 2(a)	F_1	1679.224	2.216	0
Fig. 2(b)	F_1	5.070	2.216	1.218E−004
Fig. 2(c)	F_1	182.194	2.216	4.199E−181
Fig. 2(d)	F_1	1049.810	2.216	0
Fig. 2(e)	F_1	27.748	2.216	7.066E−028
Fig. 2(f)	F_1	411.453	2.216	0
Fig. 3(a)	F_2	530.233	2.216	0
Fig. 3(b)	F_2	8.898	2.216	1.976E−008
Fig. 3(c)	F_2	44.719	2.216	1.826E−045
Fig. 3(d)	F_2	442.608	2.216	0
Fig. 3(e)	F_2	91.712	2.216	2.643E−093
Fig. 3(f)	F_2	147.058	2.216	1.107E−147
Fig. 5(a)	Alarm0.0	536.773	2.374	0
Fig. 5(b)	Alarm0.0	75.537	2.374	2.845E−062
Fig. 5(c)	Alarm0.0	149.417	2.374	6.947E−121
Fig. 5(d)	Alarm0.2	416.051	2.374	0
Fig. 5(e)	Alarm0.2	68.272	2.374	2.395E−056
Fig. 5(f)	Alarm0.2	125.232	2.374	5.001E−102
Fig. 6(a)	TSPsquare16	3634.109	2.216	0
Fig. 6(b)	TSPsquare16	3096.916	2.216	0
Fig. 6(c)	TSPsquare16	1209.700	2.216	0
Fig. 6(d)	TSPBerlin52	3351.240	2.216	0
Fig. 6(e)	TSPBerlin52	1620.826	2.216	0
Fig. 6(f)	TSPBerlin52	941.866	2.216	0

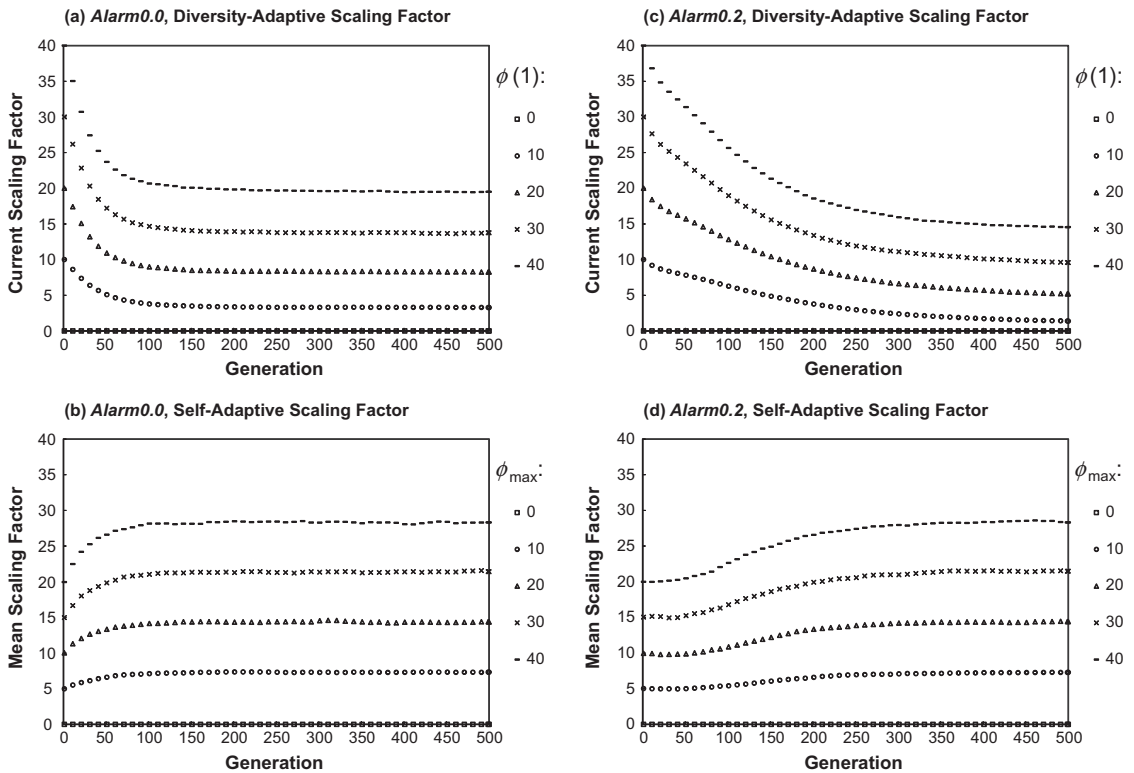

Fig. 7. Scaling-factor values averaged over 1000 runs for Alarm0.0 (left graphs) and Alarm0.2 (right graphs) with population size $M = 20$. Cases studied: diversity-adaptive scaling factor (graphs (a) and (c)), and self-adaptive scaling factor (graphs (b) and (d)). Each graph contains tests for $\phi_0 \in \{0, 10, 20, 30, 40\}$. These graphs correspond to Fig. 5b, c, e, and f.

Fig. 7 shows that the convergence speed of scaling-factor values depends on the complexity of the optimization problem; for example, for Alarm 0.0 (a "simple" problem, represented in Fig. 7a and b) convergence takes place around generation 100, whereas for Alarm 0.2 (a "complex" problem, represented in Fig. 7c and d) convergence occurs around generation 300.

By comparing the convergence values for diversity-adaptive scaling factor (Fig. 7a and c]) with those for self-adaptive scaling factor (Fig. 7b and d]), it can be observed that self-adaptation produces somewhat higher scaling-factor convergence values; for example, diversity-adaptive control for $\phi_0 = 40$ converges to a scaling-factor value of 21, whereas self-adaptive control for $\phi_0 = 40$ converges to a scaling-factor value of 27. This tendency is observed for all the cases represented in Fig. 7. Given that diversity-adaptive control slightly outperforms self-adaptive control in this case (see Fig. 5b, c, e, and f), this suggests that self-adaptive control of ϕ would benefit from applying a slightly lower degree of exploration in this domain. Nonethelesss, the similar mean best-fitness values obtained for self-adaptive and diversity-adaptive control of ϕ in all the domains studied in this work demonstrate that self-adaptation is a robust method when applied to crowding GAs.

5.5. Self-adaptation and survivor selection

Survivor selection is typically performed in crowding GAs by means of a generational method. Although Section 4 demonstrated the benefits of self-adapting the scaling factor under generational survivor selection, it is interesting to investigate the potential influence of other survivor selection methods on self-adaptive generalized crowding. In particular, we can consider other evolutionary algorithms that successfully apply self-adaptation [36,37,3,31,15].

Evolution strategies are perhaps the most widely used evolutionary approach among those that carry out self-adaptation of parameters, namely the mutation step sizes. The survivor selection mechanisms recommended in evolution strategies are the so-called $(\mu + \lambda)$ and (μ, λ) methods. The $(\mu + \lambda)$ method amounts to generating λ offspring from the μ individuals in the current population and selecting the best μ individuals from the union of the current individuals and their offspring. The (μ, λ) method is similar to $(\mu + \lambda)$, but the final selection is performed only from the λ offspring ($\lambda \geqslant \mu$ in this case).

Self-adaptation in generalized crowding can benefit from the use of (μ, λ) survivor selection method.[2] Since $\lambda \geqslant \mu$, which means that every individual can be taking part on average in λ/μ recombinations and subsequent crowding replacements, more opportunities are given to the ϕ_i value of every individual i to be tested. (Bear in mind that in generational survivor selection for crowding GAs, which is a (μ, μ) method, for each generation each individual takes part in only one recombination and crowding replacement process.) Therefore, the (μ, λ) method would provide self-adaptive generalized crowding with additional robustness regarding how the more effective ϕ-values survive in the population.

Fig. 8 depicts the behavior of self-adaptive generalized crowding under (μ, λ) survivor selection for several values of λ. In Fig. 8, Alarm 0.2 BN was used, $\mu = 20$, $\lambda \in \{20, 30, 40, 50, 60\}$, and $\phi_0 = 100$. Note that the results in Fig. 8a for fixed scaling factor are outperformed in Fig. 8b by self-adaptive scaling factor for the different λ values. Fig. 8c shows the evolution, generation by generation, of the mean scaling factor in the population over one thousand runs for the self-adaptive method; the high value chosen for $\phi_{\max}$, $\phi_{\max} = 100$, illustrates robustness and allows the mean scaling factor's changes to be clearly perceived in Fig. 8c throughout the generations. An increase in λ gives rise to a change in the corresponding mean scaling factor graph in Fig. 8c. Intuitively, as λ is increased, more states of the search space are visited in each generation and, as a consequence, less exploration is needed in the replacement phase. The self-adaptive mechanism acts in accordance with this result, as can be deduced from Fig. 8c, where higher λ corresponds to mean scaling factor graphs closer to the horizontal axis. Fig. 8c shows that the mean scaling factor in the final generation of the Alarm0.2 GA is more sensitive to λ changes when $\lambda \approx \mu$. For example, $\lambda = 20$ and $\lambda = 30$ produce quite different curves, whereas $\lambda = 50$ and $\lambda = 60$ yield almost identical curves.

5.6. Summary

One important conclusion that can be drawn from the experimental results described in Section 4 is that adaptive control of ϕ throughout the search process often produces important advantages with regards to GA performance and consistency. The control schedule for ϕ is directed by the GA itself, releasing the user from this non-trivial and problem-dependent task.

The improved consistency of the adaptive methods (consistency of their results in Table 4 when ϕ_0 is varied) is quantified in Table 6. This table shows the standard deviations of the mean best fitnesses in the final GA generation obtained in Table 4 for the different ϕ_0 values. Note that Table 6 has the same columns as Table 4 except that for ϕ_0, and that each configuration for one optimization problem and one P_M value in Table 4 is assigned one row in Table 6. The results in Table 6 quantitatively demonstrate that the adaptive methods are more consistent (give rise to smaller standard deviations) than the fixed method when ϕ_0 is varied.

It is interesting to compare the performance obtained for diversity-adaptive and self-adaptive control of ϕ in Section 4. In general, diversity-adaptive ϕ outperforms self-adaptive ϕ for the F_1, Michalewicz, Schwefel, and Alarmρ cases, whereas the opposite happens for F_2 and TSP. As a consequence, it is difficult to say which one of these two adaptive methods is best. Being comparable to diversity-adaptive control of ϕ is a remarkable result for our self-adaptive generalized crowding GA,

[2] The $(\mu + \lambda)$ survivor selection method does not fully preserve the effects of crowding, since the replacement phase may result in the worse among parent and child to survive. However, this may be subsequently undone by the $(\mu + \lambda)$ method, under which the surviving individuals are those with higher fitness. This never happens in the case of (μ, λ), where the result of the generalized crowding replacement phase is always maintained.

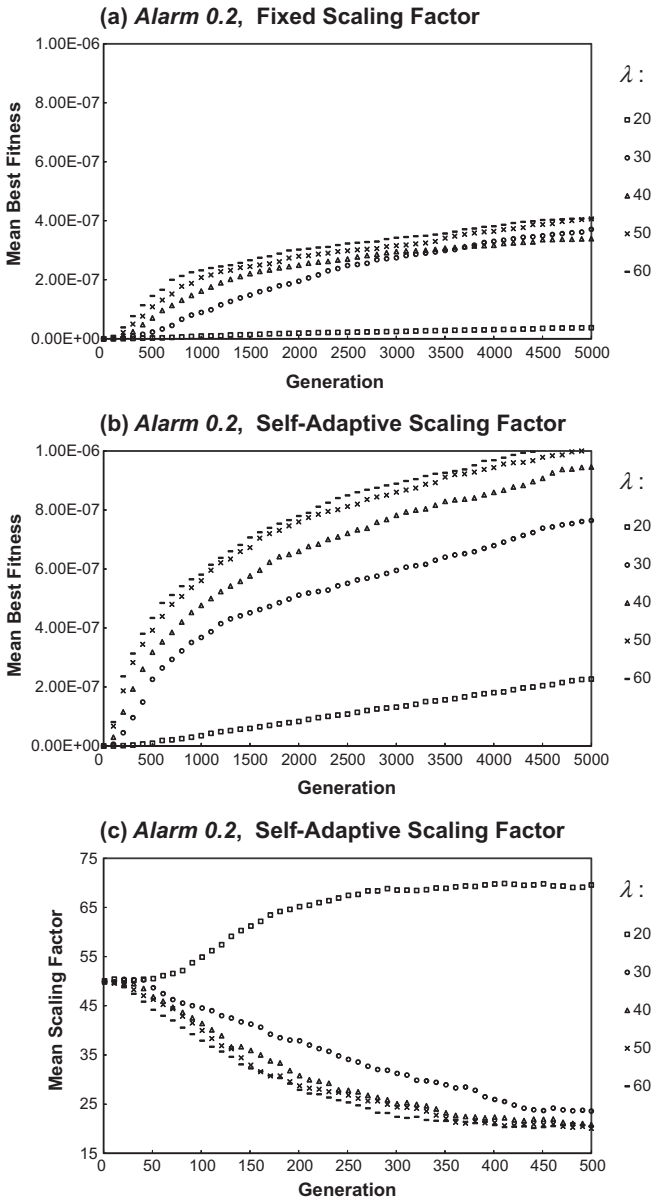

Fig. 8. Mean best fitness results over 1000 runs for Alarm0.2 under $(20,\lambda)$ survivor selection and $\phi_0 = 100$. Cases studied: (a) fixed scaling factor and (b) self-adaptive scaling factor. Each case was tested for $\lambda \in \{20, 30, 40, 50, 60\}$. Graph (c) depicts the evolution of the mean scaling factor in the population for the self-adaptive case (b).

Table 6
Standard deviations of the mean best fitnesses in the final GA generation obtained in Table 4 for the different ϕ_0 values.

Optimization problem	P_M	Standard deviation Scaling-factor control methods:		
		Nonadaptive (fixed)	Diversity-adaptive	Self-adaptive
F_1	0.0125	0.1183	0	0.00084
	0.025	0.06084	0	0.00082
F_2	0.0125	0.05689	0.01318	0.01606
	0.025	0.04004	0.02567	0.02008
Alarm0.0	$\frac{1}{37}$	1.84E−8	9.46E−9	1.21E−8
Alarm0.2	$\frac{1}{37}$	1.36E−6	6.46E−7	8.03E−7
TSPsquare16	0.6	204.33	192.54	98.39
TSPBerlin52	0.6	1021.25	761.47	549.18

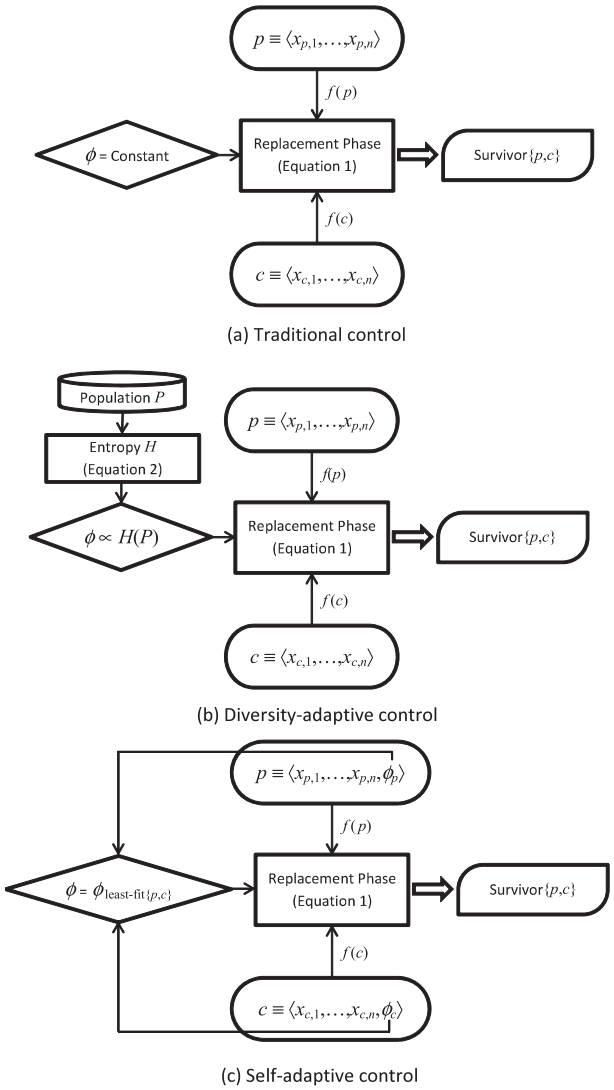

Fig. 9. Operation of the traditional (a), diversity-adaptive (b), and self-adaptive (c) control methods during the replacement phase of generalized crowding.

since diversity-adaptive control of ϕ takes advantage of heuristic knowledge establishing that the degree of exploration should be proportional to the diversity of the current population.

Although this article addresses the application of adaptive generalized crowding techniques in the context of GAs, many population-based search algorithm can take advantage of these techniques. The only requirement is that, at each iteration of the algorithm, a population of individuals is used to generate new candidate individuals. Specifically, adaptive generalized crowding techniques can be applied to the set of current and candidate individuals in order to determine which individuals will survive. Some examples of other population-based algorithms where generalized crowding techniques can be applied are the following:

1. In differential evolution [39,6], for each individual x in the population a potential new individual y is generated from the recombination of x and an intermediate individual z calculated from a simple formula. If y is more adapted than x, y replaces x in the population. Instead of this deterministic rule for the competition between x and y, the different replacement rules discussed in this article could be applied.
2. In particle swarm optimization [21,35], the position of each particle in the swarm is normally updated at each iteration depending on its velocity, its local best known position, and the best known position in the search space. Instead of always updating the particle's position, the replacement rules discussed in this article could be used to decide whether or not the particle is moved to its next calculated position.

3. In simulated annealing [22,1], after picking some initial candidate solution, a new neighboring candidate is chosen at each iteration. The neighboring candidate is evaluated and possibly replaces the current candidate after the application of the Metropolis criterion. Instead of this criterion, any of the replacement rules described in this article could be employed.

6. Conclusion and future work

Generalized crowding is used in GAs as a technique for preventing premature convergence and keeping diversity in the population. This is done by means of a scaling factor parameter, which traditionally is kept fixed throughout search [14]. Determining the proper value for the scaling factor is a tedious and problem-dependent task. In order to solve this problem, this work introduces two techniques for adapting the scaling factor: diversity-adaptive control and self-adaptive control. Fig. 9 summarizes how the proposed methods work, compared to the traditional method, during the replacement phase of crowding. Whereas traditional generalized crowding uses a fixed scaling factor, the diversity-adaptive method employs information on the entropy of the current population to update the scaling factor, and the self-adaptive method incorporates the scaling factor into the chromosomes of individuals.

The experiments illustrate, in many cases, a benefit when the scaling factor is adapted compared to when it is kept fixed. As summarized in Table 4, the two new techniques give, compared to the original fixed scaling factor method, comparable or better results for solution quality in most of the experiments, and also have better consistency.

With regards to which adaptive control method is best, the experiments show that there is no clear winner between diversity-adaptive and self-adaptive control. Specifically, diversity-adaptive ϕ outperforms self-adaptive ϕ for the F_1, Michalewicz, Schwefel, and Alarmρ cases, whereas the opposite happens for F_2 and TSP. Further experimental studies may determine under what conditions and to what extent one of the methods outperfoms the other and vice versa.

Although we mainly used generational survivor selection in our experiments, we also established in Section 5 that the survivor selection method (μ, λ) adapts well to self-adaptive generalized crowding. A more detailed study of the (μ, λ) method in the context of self-adaptive generalized crowding could be performed as future research.

Finally, this work puts emphasis on global optimization through generalized crowding GAs, rather than on analyzing niche formation for adaptive methods. The latter problem would be interesting to investigate in future research.

References

[1] E. Aarts, J. Korst, W. Michiels, Simulated annealing, in: E.K. Burk, G. Kandall (Eds.), Search Methodologies – Introductory Tutorials in Optimization and Decision Support Tecniques, Springer, New York, 2005, pp. 187–210.
[2] D.H. Ackley, A Connectionist Machine for Genetic Hillclimbing, Kluwer Academic Publishers, Boston, MA, 1987.
[3] T. Bäck, Self-adaptation in genetic algorithms, in: F.J. Varela, P. Bourgine (Eds.), Proceedings of the First European Conference on Artificial Life, The MIT Press, 1992, pp. 263–271.
[4] P.J. Ballester, J.N. Carter, Real-parameter genetic algorithms for finding multiple optimal solutions in multi-modal optimization, in: E. Cantú-Paz, J.A. Foster, K. Deb, L.D. Davis, R. Roy, U. O'Reilly, H. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M.A. Potter, A.C. Schultz, K.A. Dowsland, N. Jonoska, J. Miller (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003), Springer-Verlag, 2003, pp. 706–717.
[5] I.A. Beinlich, H.J. Suermondt, R.M. Chavez, G.F. Cooper, The ALARM monitoring system: a case study with two probabilistic inference techniques for belief networks, in: J. Hunter, J. Cookson, J. Wyatt (Eds.), Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine (AIME'89), Springer, 1989, pp. 246–256.
[6] U.K. Chakraborty (Ed.), Advances in Differential Evolution, Springer, Berlin Heidelberg, 2008.
[7] A. Darwiche, Modeling and Reasoning with Bayesian Networks, Cambridge University Press, 2009.
[8] L. Davis, Applying adaptive algorithms to epistatic domains, in: Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85), Morgan Kaufmann, 1985, pp. 162–164.
[9] K.A. de Jong, An analysis of the behavior of a class of genetic adaptive systems, PhD thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI, 1975.
[10] A.E. Eiben, C.A. Schippers, On evolutionary exploration and exploitation, Fundamenta Informaticae 35 (1998) 1–16.
[11] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Springer, Berlin, 2003.
[12] A. Farhang-Mehr, S. Azarm, Entropy-based multi-objective genetic algorithm for design optimization, Structural and Multidisciplinary Optimization 24 (5) (2002) 351–361.
[13] S.F. Galán, O.J. Mengshoel, Constraint handling using tournament selection: abductive inference in partly deterministic Bayesian networks, Evolutionary Computation 17 (1) (2009) 55–88.
[14] S.F. Galán, O.J. Mengshoel, Generalized crowding for genetic algorithms, in: M. Pelikan, J. Branke (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010), ACM Press, 2010, pp. 775–782.
[15] S.F. Galán, O.J. Mengshoel, R. Pinter, A novel mating approach for genetic algorithms, Evolutionary Computation 21 (2) (2013) 197–229.
[16] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[17] D.E. Goldberg, R. Lingle, Alleles, loci, and the traveling salesman problem, in: J.J. Grefenstette (Ed.), Proceedings of the 1st International Conference on Genetic Algorithms (ICGA-85), Lawrence Erlbaum Associates, 1985, pp. 154–159.
[18] G.R. Harik, Finding multimodal solutions using restricted tournament selection, in: L.J. Eshelman (Ed.), Proceedings of the 6th International Conference on Genetic Algorithms (ICGA-95), Morgan Kaufmann, 1995, pp. 24–31.
[19] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, MI, 1975. Second edition, The MIT Press, 1992.
[20] F.V. Jensen, Bayesian Networks and Decision Graphs, Springer, Berlin, 2001.
[21] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks IV, IEEE Press, 1995, pp. 1942–1948.
[22] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[23] P. Larrañaga, C. Kuijpers, R. Murga, I. Inza, S. Dizdarevich, Evolutionary algorithms for the travelling salesman problem: a review of representations and operators, Artificial Intelligence Review 13 (1999) 129–170.
[24] S.H. Liu, M. Mernik, B.R. Bryant, To explore or to exploit: an entropy-driven approach for evolutionary algorithms, International Journal of Knowledge-Based and Intelligent Engineering Systems 13 (2009) 185–206.

[25] S.W. Mahfoud, Crowding and preselection revisited, in: R. Männer, B. Manderick (Eds.), Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature (PPSN II), Elsevier, 1992, pp. 27–36.
[26] S.W. Mahfoud, Niching methods for genetic algorithms, PhD thesis, Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1995.
[27] S.W. Mahfoud, D.E. Goldberg, Parallel recombinative simulated annealing: a genetic algorithm, Parallel Computing 21 (1995) 1–28.
[28] O.J. Mengshoel, Efficient Bayesian network inference: genetic algorithms, stochastic local search, and abstraction, PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.
[29] O.J. Mengshoel, D.E. Goldberg, Probabilistic crowding: deterministic crowding with probabilistic replacement, in: W. Banzhaf, J.M. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M.J. Jakiela, R.E. Smith (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999), Morgan Kaufman, 1999, pp. 409–416.
[30] O.J. Mengshoel, D.E. Goldberg, The crowding approach to niching in genetic algorithms, Evolutionary Computation 16 (3) (2008) 315–354.
[31] S. Meyer-Nieberg, H.G. Beyer, Self-adaptation in evolutionary algorithms, in: F.G. Lobo, C.F. Lima, Z. Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, Springer, Berlin, 2007, pp. 47–75.
[32] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, third ed., Springer, Berlin, 1996.
[33] H. Mühlenbein, D. Schomisch, J. Born, The parallel genetic algorithm as function optimizer, Parallel Computing 17 (1991) 619–632.
[34] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufman, San Francisco, CA, 1988. Revised second printing, 1991.
[35] R. Poli, J. Kennedy, T. Blackwell, Particle swarm optimization, Swarm Intelligence 1 (1) (2007) 33–57.
[36] I. Rechenberg, Cybernetic solution path of an experimental problem, Royal Aircraft Establishment (1965). Library translation 1122.
[37] H.P. Schwefel, Numerical Optimization of Computer Models, Wiley, New York, 1981.
[38] E. Shimony, Finding MAPs for belief networks is NP-hard, Artificial Intelligence 68 (1994) 399–410.
[39] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341–359.
[40] G. Syswerda, Uniform crossover in genetic algorithms, in: J.D. Schaffer (Ed.), Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA-89), Morgan Kaufman, 1989, pp. 2–9.
[41] G. Syswerda, Schedule optimization using genetic algorithms, in: L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991, pp. 332–349.
[42] Y. Tsujimura, M. Gen, Entropy-based genetic algorithm for solving TSP, in: L.C. Jain, R.K. Jain (Eds.), Proceedings of the 2nd International Conference on Knowledge-Based Intelligent Electronic Systems (KES'98), IEEE Press, 1998, pp. 285–290.
[43] D. Whitley, T. Starkweather, D. Fuquay, Scheduling problems and travelling salesman: the genetic edge recombination operator, in: J.D. Schaffer (Ed.), Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA-89), Morgan Kaufman, 1989, pp. 133–140.
[44] J. Wroblewski, Theoretical foundations of order-based genetic algorithms, Fundamenta Informaticae 28 (3–4) (1996) 423–430.