INFORMS TutORials in Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Nature-Inspired Heuristics: Overview and Critique

Craig A. Tovey

To cite this entry: Craig A. Tovey. Nature-Inspired Heuristics: Overview and Critique. *In* INFORMS TutORials in Operations Research. Published online: 19 Oct 2018; 158-192.
https://doi.org/10.1287/educ.2018.0187

https://pubsonline.informs.org/series/educ/

TUTORIALS IN OPERATIONS RESEARCH
 | ISBN 978-0-9906153-2-3
https://doi.org/10.1287/educ.2018.0187

Nature-Inspired Heuristics: Overview and Critique

Craig A. Tovey
H. Milton School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332
Contact: cat@gatech.edu, http://orcid.org/000-0002-1274-909X (CAT)

Abstract Evolutionary programming, genetic algorithms, simulated annealing, ant colony optimization, and particle swarm optimization are among the earliest optimization heuristics inspired by animate and inanimate phenomena in the natural world. Some of the more recently invented methods have exotic names such as roach infestation, shuffled frog-leaping, invasive weed optimization, and cuckoo search. This tutorial is a guide to the bewildering, burgeoning menagerie of such heuristics, which now comprises more than 100 algorithms, and whose publications number in the hundreds of thousands. Their underlying principles include populations, recombination, exploration, reinforcement, encoding, selection, randomness, and perturbation. They have been successful in many implementations, often winning out against classical operations research (OR)/computer science methods for implementation or a user's acceptance. They have been less successful, sometimes spectacularly so, in many computational test comparisons with classical methods. I speculate as to why these success levels differ so greatly. On the one hand, I critique the insularity and mathematical naiveté of this heuristic research, particularly its limited nature-versus-nature comparisons and self-contradictory stance on algorithm parameter tuning. On the other hand, I critique the OR community's having implicitly ceded important optimization territory to others and our failure to face what is now there. In an attempt to spur our community to thoroughly engage with nature-inspired heuristics, I conclude by observing a misalignment between individual and community incentives, identifying a few potentially powerful nature-inspired heuristic ideas for optimization, and proposing some specific research questions that may attract operations researchers.

Keywords swarm intelligence • simulated annealing • heuristic • optimization • genetic algorithm • particle swarm optimization • evolutionary algorithm • ant colony optimization • bee algorithm • differential evolution • swarm intelligence • firefly algorithm • cuckoo search • biomimicry • biologically inspired • nature-inspired

1. Introduction

Inspiration from nature begins when we observe something in the natural world that is better, in some way, than human-built things with the same functionality. We next study the natural phenomenon to understand how and why it works well. Then we apply our new understanding to build something with better functionality than we have achieved previously. Velcro™, invented by George de Mestral after he examined why seeds stuck to his dog's wet fur, is the prototypical example. There are many successful examples of such inspiration in architecture (Pawlyn [90]), robotics (Pfeifer et al. [91], Quinn et al. [96]), and other fields (Benyus [8], Center for Biologically Inspired Design [18]). This tutorial is about *nature-inspired heuristic optimization algorithms*, or NIAs for short. Simulated annealing and genetic algorithms are two of the earliest NIAs, both widely known since the mid-1980s. Several more basic algorithms were invented in the early 1990s. But nothing augured the explosive growth of the field from the mid-1990s to the present. Here are the main points of this tutorial:

- NIAs are randomized iterative search heuristics based on analogies from nature. Most repeatedly update a set of candidate solutions; a few update a single candidate. They offer a continuum of trade-offs between CPU time and solution quality. They have no deterministic guarantee of finite convergence, nor do they obtain any lower bounds on solution quality.
- Among the biologically inspired algorithms, two analogies predominate: *evolutionary* and *physical movement*. According to an evolutionary analogy, the set of candidate solutions is a population, and the updated set comprises that population's offspring. According to a physical movement analogy, particles move in the search space. Particles may represent bees, whales, wolves, ants, etc. The set of candidate solutions consists of the particles' positions. The particles move to new positions each iteration. The updated set of solutions is the set of new positions. The analogies differ greatly, but much of the underlying mathematics does not.
- NIAs generate new candidate solutions in novel ways by combining information from previous ones. By analogy to the "survival of the fittest" principle, the better a solution's objective value, the more influence it has on new candidates.
- Survival of the fittest focuses search in regions where good-quality solutions have already been found. This principle is called exploitation or intensification. To avoid premature local convergence, most NIAs employ the counterbalancing principle to search in regions that have not yet been searched. This principle is called exploration or diversification.
- NIA research tends to be insular, comparing computational results only with other NIA results. The ostensible reason is that it is not fair to compare general-purpose NIAs with algorithms designed for specific problems. However, to apply an NIA to a specific problem, one must make critical design decisions and then perform extensive tuning on the algorithm's parameters.
- Much but not all NIA research has been at low technical levels of mathematics, numerical computation, and/or computational complexity. Terms such as "solve" and "optimize" have different meanings than in operations research (OR). That proving optimality is not in the same complexity class as is finding an optimal solution goes unmentioned.
- NIA research has exploded since the mid-1990s. Approximately 20,000 NIA papers were published during the past 12 months, compared with roughly 12,000 in linear, integer, conic, and nonlinear programming and combinatorial optimization combined. On well-studied OR problems such as the shortest-path problem and the traveling salesman problem (TSP), NIAs perform worse by factors of 10^3 to 10^6 or more, notwithstanding their not guaranteeing any solution quality. Moreover, NIAs to date do not scale up well to larger instances of more than a hundred variables. We in the OR community have not adequately communicated this disparity to scientists and engineers.
- According to my experience and observation, the closer the analogy between what nature accomplishes remarkably well and what we want to accomplish, the more valuable the lesson from nature. Despite my preceding wholesale critiques of the NIA field, it has some excellent ideas that we in OR should explore and incorporate. These include searching in a space of algorithm parameters, algorithms, or instances, rather than the space of solutions; and pooling information from different good solutions to find a potentially better solution or set of high-quality solutions. We could be making far more use of NIAs for decentralized control of autonomous robots or other agents, to generate sets of broadly varied Pareto-optimal solutions for multiobjective problems, and for high-level repeated use of classical methods.
- More generally, I urge the OR community to fully engage with the NIA and "metaheuristic" communities. If we do not, what we have perhaps dismissed as "invasive weeds" threatens to supplant us in much of our methodological and applications territory. To engage successfully, we ought to acknowledge that we have deemphasized topics and methods that do not admit of "nice" mathematical results, and remember to be objectively corrective but not contemptuous.

2. Concepts and Terminology

2.1. Core Optimization Problems

The field of nature-inspired algorithms addresses both continuous and discrete optimization. The field's most common continuous optimization problem is what we in OR call unconstrained global optimization:

$$\min_{x \in \Re^n} f(x).$$

Any $x \in \Re^n$ is a *solution*; $f(x)$ is its *objective value*.

Functions $f(x)$ with multiple local optima get considerably more attention than convex or other functions with a single local optimum. The ability to escape local optima is considered one of the chief advantages of NIAs. Constrained optimization is usually limited to, in decreasing order of frequency, (i) constraints treated as penalty costs, (ii) box constraints (upper and/or lower bounds on variables), and (iii) linear inequality constraints. In general, when feasible regions are not convex, NIAs for continuous optimization have trouble ensuring feasibility of new candidate solutions. This is because mixtures of feasible solutions are apt to be infeasible. Thus, NIAs are rarely used for continuous optimization subject to general nonlinear inequality constraints.

The field's most common discrete optimization problem is unconstrained minimization on the set of binary strings of length n,

$$\min_{x \in \{0,1\}^n} f(x).$$

Any $x \in \{0,1\}^n$ is a *solution*; $f(x)$ is its *objective value*. The jth component of x is denoted x_j. The Hamming distance $H(x, y) \equiv \sum_{j=1}^{n} |x_j - y_j|$ is the standard metric on the set of solutions. As with continuous optimization, NIAs can have trouble retaining feasibility in constrained discrete optimization. Instead of shying away from such problems, NIA researchers have attacked them directly, thereby accounting for a large portion of the NIA literature. I think there are two key reasons for the greater attention paid to constrained discrete optimization. First, global optimization is difficult without constraints, but discrete optimization is usually meaningless or trivial without constraints. Second, constraints in discrete optimization problems such as the TSP restrict the feasible region according to a highly structured pattern. The resulting structure of the feasible region is easier to handle than the irregular disconnected nonconvex regions of constrained global optimization.

Many search algorithms, as they proceed, store the *incumbent*, a best solution found so far. Its objective value is the *incumbent value*, which by definition is the minimum objective value of all solutions the algorithm has evaluated.

Some NIAs have their own vocabularies for standard mathematical terms. For instance, in a genetic algorithm, variables are called "genes," variable values such as the integers 0 and 1 for binary variables are called "alleles," and solutions are called "chromosomes." As another example, retention of the incumbent is called "elitism." I avoid these terms when I describe the algorithms. They have an exotic analogical allure but are not necessary to understand the content.

On the other hand, NIAs employ several concepts not standardly used in the OR field of optimization. These concepts require terminology new to many OR readers. I define them next.

2.2. Populations

"Why is there sex?" This is a core question in evolutionary biology. From a selfish gene's point of view, why risk a 50% chance of not being replicated in an offspring, versus the nearly certain replication of asexual reproduction? The basic answer is that sexual reproduction confers far more adaptiveness because the progeny have different combinations of traits and hence different levels of fitness. Sex greatly increases the pace of evolutionary adaptation.

In my view, the primary contribution of nature-inspired heuristic algorithms has been to incorporate sex into optimization. In mathematical terms, the most distinctive concept of NIAs is to work simultaneously with many solutions that interact with each other. By contrast, classical optimization search algorithms typically work with one solution at a time.

A *population* X is a multiset of solutions; that is, each member of X is a solution, and the members of X need not be distinct. Each $x \in X$ may be accompanied by ancillary data denoted by $\mathscr{x}$; ancillary data for the entire population X are denoted by $\mathscr{X}$. For a prototypical example, suppose the incumbent $z \notin X$. Unless the implementation is very wasteful, $z \notin \mathscr{x}$ for any $x \in X$. The algorithm would store z and its objective value as ancillary data in $\mathscr{X}$.

2.3. The General Form of an NIA

On a high level, most NIAs fit the following general framework.

Algorithm 1 (Basic Nature-Inspired Algorithm Framework to Minimize $f(x)$)

Step 0. Choose an initial population X and ancillary data $\mathscr{X}$.
Step 1. Evaluate $f(x)$ for all $x \in X$.
Step 2. Terminate if stopping criterion is met.
Step 3. Build a population Y and its ancillary data $\mathscr{Y}$ based on X, f, and $\mathscr{X}$.
Step 4. Set $X = Y$, set $\mathscr{X} = \mathscr{Y}$, and go to Step 1.

A few clarifications are in order.

- Step 3 is always randomized with probabilities favoring the use of solutions in X with better objective values.
- The stopping criterion is usually an upper bound on computation time or iterations, either in total or since an appreciable improvement in incumbent value has occurred.
- The four most notable features are (i) population, (ii) randomization, (iii) interactions among population members, and (iv) oracular "black box" use of the objective function f, as does simulation optimization.
- Local search fits this framework in the extreme case $|X| = 1$, except that Step 3 need not be randomized. However, many NIAs are surprisingly poor at reaching a local optimum, especially in continuous optimization. I would augment them with an intermediate step:

  Step 3.5. Update Y by performing local search from each $y \in Y$.
- By an analogy with evolution, the population Y built in Step 3 may be called a sequence of *generations*. By an analogy on a much smaller time scale, each population may be considered a configuration $X, \mathscr{X}$ of a group of mobile organisms whose movements lead to the next configuration, $Y, \mathscr{Y}$.

Step 3 is obviously the core of the basic NIA. The NIA literature abounds in different ways to build the next population Y from the current population X. Many of these have their own names and biological analogies. This plethora can bewilder outsiders. To simplify the exposition, I artificially separate the creation of a population into two parts, *solution generation* and *selection*. Solution generation normally consists of operations that, given one member $v \in X$ or two members $v \in X, w \in X$ of the population, their ancillary data $\mathscr{v}$ or $\mathscr{v}, \mathscr{w}$, and the population ancillary data $\mathscr{X}$, produce solutions that are possible members of the next population Y. Selection chooses which of those solutions will compose Y. This separation is algorithmically false. It implies populations are built in two steps, solution generation followed by selection. But it would be idiotic to generate exponentially many possible solutions from which to select a handful. Instead, NIAs may apply selection criteria both before and after solution generation. An NIA may select which members of X are the input to the generation operations. After the possible members of Y have been generated, the NIA may select which will compose Y. Following the evolutionary metaphor, fitness corresponds to the number of progeny; it entails both reproduction and survival.

Figure 1. Crossovers of x and y combine the prefix of one with the suffix of the other.

$$x = 111000111000111000 = \overbrace{11100011100}^{m} \circ \overbrace{0111000}^{n-m} \qquad z = 111000111001010101$$

$$\Rightarrow$$

$$y = 010101010101010101 = \overbrace{01010101010}^{m} \circ \overbrace{1010101}^{n-m} \qquad z' = 010101010100111000$$

2.4. Three Methods to Generate Solutions from a Population

In my view, NIAs employ three main conceptual ways to construct potential members of the next population Y from the present one X. In this tutorial I call them perturbation, recombination, and attraction. These concepts and their variations go under various names in the NIA literature.

Perturbation of a solution x is a randomized alteration of x, standardly performed independently on each component x_j. In discrete optimization, the simplest perturbation employs one parameter $0.5 > \epsilon > 0$. For each $j = 1, \ldots, n$, change x_j to $1 - x_j$ with probability ϵ, independent of other j. In continuous optimization, the simplest perturbations employ one parameter $\epsilon > 0$. They generate n independent and identically distributed random values $\delta_j \sim \mathrm{U}[-\epsilon, \epsilon]$ and alter x to $x + \delta$. In variations of this perturbation, the δ_j can have Gaussian distributions to make δ have a spherically symmetric distribution, or ϵ can be drawn from a probability distribution instead of being a fixed parameter. In genetic and other algorithms based on the analogy to evolution, perturbation is called "mutation," the biological term for random changes to DNA that are passed to progeny. For this tutorial I prefer the more mathematically descriptive term "perturbation."

In most NIAs, ϵ is small, so with high probability, the perturbed solution is close to x. In a few NIAs, ϵ is drawn from a heavy-tailed distribution, resulting in the occasional generation of a new solution quite distant from x. I explain the reason for this in Section 2.6.

Perturbation is not a radically new concept for us in OR because it is similar to random movement within a neighborhood of a solution. *Recombination*, however, is radically different from local search. The general idea is to generate randomized new solutions by combining two existing solutions, x and y. The simplest such operation is called single point *crossover*, illustrated in Figure 1.

Algorithm 2 (Single-Point Crossover of Solutions x and y)

Step 1. Treat x and y as strings of length n. Define the first (respectively, last) m terms of a string w as the prefix $P_m(w)$ (respectively, suffix $S_m(w)$). Thus if w has length n, it is the concatenation $w = P_m(w) \circ S_{n-m}(w)$.

Step 2. Generate a random integer m, called the *crossover point* or *site*, in the range $0 \le m \le n$.

Step 3. The two offspring of x and y are $z = P_m(x) \circ S_{n-m}(y)$ and $z' = P_m(y) \circ S_{n-m}(x)$.

A more general crossover operation has k random crossover points $m_1 \le m_2 \le \cdots \le m_k$. Randomly starting with x or y, it builds z by concatenating pieces taken alternately from x and y. The first piece comprises components $m_0 \equiv 1, \ldots, m_1$; the ith piece comprises components $1 + m_{i-1}, \ldots, m_i$; and the last piece comprises components $1 + m_k, \ldots, m_{k+1} \equiv n$. Various other recombination operations have been invented for different problems. For instance, one can set z_i randomly to either x_i or y_i with equal probability, independently for each i. In a more general version of this operation, create two offspring z and z'. For parameter value $p \ge 1/2$, set $z_i = x_i$ with probability p and $z_i = y_i$ otherwise; independently set $z'_i = y_i$ with probability p and $z'_i = x_i$ otherwise. Crossover as defined here mimics a pair of chromosomes splitting and merging. In practice, crossover can be so altered for the sake of speed or solution quality that its metaphorical origin is imperceptible, yet it is still called "crossover." That is why, in the literature, "recombination" and "crossover" are frequently synonymous.

The term "recombination" comes from genetic algorithms and is standard in that literature. The third term, "attraction," is my choice for this exposition. It is one of several in the literature, none of which has emerged as dominant. The idea of attraction is to alter a solution x to make it more like a better solution y. For example, in the case of continuous variables, set z to the convex combination of x and y, $z = \theta y + (1-\theta)x$. In the case of discrete variables, initially set $z = x$. Then randomly choose some indices i such that $x_i \neq y_i$, and set $z_i = y_i$. In both cases, replace x with z.

Recombination and attraction are so similar that I almost decided not to define attraction separately. Both generate new solutions by mixing a pair of existing solutions x and y. The recombination example I gave for discrete variables that sets z_i randomly to equal x_i or equal y_i would serve equally well as an attraction example that alters x to be more like y. In this tutorial I distinguish between them conceptually because attraction can also operate on a solution x without another specific solution y. The idea is to maintain ancillary data in $\mathcal{X}$ on components of the highest-quality solutions found so far, seeking high correlations between component value and objective value.

Example 1 (Discrete Case). For some K, maintain the average v of the K best solutions found so far, or of the most recent K incumbents. That is, v_j equals the fraction of the high-quality solutions y for which $y_j = 1$. For some fairly small $\delta > 0$ such as 0.2, identify indices j for which $v_j \leq \delta$ or $v_j \geq 1-\delta$. Attract solution x to the high-quality solutions by setting $x_j = \lfloor v_j + 0.5 \rfloor$ for a random subset of those identified indices.

Example 2 (Continuous Case). Maintain both the sample mean and variance of each component of the K high-quality solutions. For indices j for which the sample variance is small, let x_j be attracted to the sample mean of component j.

Keep in mind that recombination and attraction are not very distinct algorithmically. They started from the different metaphorical bases of evolution and animal movement. Historically, their most characteristic mixing operations differed—evolution suggested crossover of x with y; physical movement suggested changing the location or velocity of x with respect to y in space. Steered by computational testing and influenced by each other, they have just about merged into one collection of procedures.

2.5. Selection

The three main principles for selection are objective value, exploration, and randomization.

Objective Value. As this principle is at the core of local improvement algorithms, it may seem so obvious as to require no exposition. Please bear with me.

First, in evolution-inspired NIAs, the objective value is usually called *fitness*. This is by analogy to the biological meaning of fitness, the expected number of progeny. However, besides the trivial discrepancy by a factor of -1 when minimizing an objective function, the analogy is inexact. There is no definitive way to normalize objective values. If animal x has twice the fitness of y, we should simulate reproduction so that x has twice the expected number of progeny as y. Suppose, for instance, $f(x) = 50$ and $f(y) = 100$ in a minimization problem. We could plausibly rate x to be twice as good as y if the optimum objective value were 1. But if the optimum value were 49 (respectively, -10^3), it would be more plausible to rate x's fitness to be 51 (respectively, $1{,}100/1{,}050 \approx 1.05$) times y's fitness. One class of solutions to this normalization conundrum uses the range of objective values in the population as a yardstick. Another class uses only the ordinal objective values.

Second, in the NIA literature, this principle is most often called *exploitation* or *intensification*. These terms reflect the notion that the algorithm has found a region in the search space that contains high-quality solutions and therefore should exploit that region by intensifying its search activity there. As is often the case in NIAs, there is a frustrating multiplicity of

terminology for this principle. Other terms that are moderately popular are *reinforcement* and *refinement*.

Third, there are many ways to favor the solutions with better objective values from among a population Z. Some of the most common selection rules in NIAs are as follows:

- Select the solutions with the $\alpha|Z|$ best objective values for some $\alpha < 1$. The value of α is often set to $\frac{1}{2}$ or $n/|Z|$, where n is the desired size of the next generation's population.
- Model the optimization problem as the maximization of a nonnegative function f. Select solution w from Z with probability $f(w)/\sum_{z\in Z} f(z)$. Selection may be done with or without replacement.
- Model the optimization problem as the minimization of a function f. Let $f_{\max} = \max_{z\in Z} f(z)$. For some $\beta > 1$, select w from Z with probability

$$\frac{\beta f_{\max} - f(w)}{\sum_{z\in Z} \beta f_{\max} - f(z)}.$$

Selection may be done with or without replacement.
- For $m << |Z|$, select the best of m randomly sampled members of Z. Sampling and selection may be done with or without replacement. This is called *tournament selection* in the genetic algorithms literature. To my mind, it is a simple way to select based on ordinal objective values, dressed up with another metaphor.

If bettering the objective value were the only selection criterion, algorithms would always get stuck at local optima. *Exploration* is the selection of solutions from regions from which few or no solutions have already been examined. This principle is complementary to favoring good objective values. The idea is to widen the scope of the search so as not to miss a distant global optimum. The term "exploration" is often alliteratively paired with "exploitation." The term *diversification*, which has been used as a synonym for exploration, is often rhymingly paired with "intensification."

Randomization, too, can enable an algorithm to escape from a local optimum. Many researchers, if their NIA does not do enough exploration, add a step that generates new random solutions with positive probability. More subtly, perhaps, randomization enables NIAs to offer time/quality trade-offs. For discrete optimization, at one extreme, deterministic local improvement takes little time on average (Aarts et al. [1], Tovey [119, 120]) but usually obtains solutions of limited quality. Randomized NIAs, similar to repeated randomized local search, give better-quality solutions as they are allotted more time. At the other extreme, sufficiently randomized NIAs reach a global optimum with probability 1 as run time goes to infinity (see Section 7.1).

Regrettably, the NIA literature uses the term "stochastic" instead of "randomized." It is merely annoying that when we read "stochastic algorithm" we have to think "randomized algorithm"; it is downright confusing that when we read "stochastic optimization algorithm" we have to think "randomized algorithm for deterministic optimization."

2.6. Premature Convergence, Genetic Drift, and Diversity Maintenance

It is a truth universally acknowledged among researchers that every NIA is in need of a balance between exploitation and exploration (Blum and Roli [11]). Insufficient exploration tends to make an algorithm converge prematurely before the globally best portions of the solution space have influenced the population; overmuch exploration is too much like random sampling to have a high probability of finding a good solution in large search space.

Premature convergence to a globally poor local optimum is nearly synonymous with *drift*. The evolutionary analogy to genetic drift is close and illuminating. Biologists often observe a genetic distribution peculiar to a small isolated population, different from and less varied than the overall distribution in the species. An allele, even if conferring a slight fitness advantage over other alleles of the same gene, can drop out of the gene pool because of vagaries of organism survival and reproductive success and because of the randomness of meiosis. If the

population remains isolated, its genetic variety remains diminished. The smaller the population, the more likely are such chance events of genetic fixation. Similarly, in an NIA, randomness as to which candidates become parents, and as to which progeny are altered in a particular location by perturbation, recombination, or attraction, can make a pattern in a substring vanish from the population.

Example 3 (Drift). For a problem on a graph containing a cycle on the vertex subset $0, 1, 2, 3, 4, 5$, let $x_i = 1 : 0 \le i \le 5$ correspond to using the edge $(i, i+1 \bmod 6)$. Let the problem require a matching on those vertices. Suppose the best two such matchings are $M1 = \{(0,1), (2,3), (4,5)\}$ and its complement, $M2 = \{(1,2), (3,4), (5,0)\}$, such that solutions having neither $M1$ nor $M2$ have relatively poor objective values.

Let the solution length $n = 100$ and the population size $|X| = 100$. Suppose the NIA generates pairs for the next generation by randomly selecting two solutions (with replacement) at or above the median solution quality and performing single-point crossover at a random site $1 \le m \le |X|$. When we begin observing the algorithm, a fraction q of the members of X above the median in quality have $M1$, and a fraction $1 - q$ have $M2$.

For each pair z, z' generated, independent of other pairs, the probabilities are

- q^2 that both have $M1$, inherited;
- $(1-q)^2$ that both have $M2$, inherited;
- $2q(1-q)P(m \ge 6) = 2q(1-q)(95/n) = 1.8q(1-q)$ that one has $M1$ and one has $M2$, inherited; and
- $2q(1-q)(5/n) = 0.1q(1-q) \le 0.025$ that both lack $M1$ and $M2$.

Let $Z \sim \text{Bin}(0.025, 50)$. By the central limit theorem, Z is approximately Gaussian with mean 1.25 and standard deviation $\sigma_Z = \sqrt{(1.25)(0.975)} \approx 1.104$. The probability that at least half of the generated solutions have $M1$ or $M2$ is at least $1 - P(Z \ge 26) \approx \Phi(24.75/\sigma_Z) < \Phi(22)$, which is absurdly close to 1. Hence we may assume that all parents always have $M1$ or $M2$.

Let $Z1$ be the number of children generated now that have $M1$ and will be better than median in the next generation. Assuming neither $M1$ nor $M2$ confers an advantage over the other, $E[Z1] = [(25)(2q^2 + 1.8q(1-q))]/[q^2 + (1-q)^2 + 1.8q(1-q)]$. A little algebra shows that $E[Z1] \ge 50q$ if $q \ge 0.5$, with equality only at $q = 0.5$ and $q = 1$, and a little arithmetic shows that $\sigma_{Z1} \ge 1$ for $0.5 \le q \le 0.94$, and $\sigma_{Z1} \ge 0.5$ for $q \le 0.98$. Therefore if $Z1 \ge 25$, it randomly walks with upward drift and an expected step size greater than or equal to 1, except for a slightly smaller size $Z1 = 48$ and 49. Of course, when $q < 0.5$, it is $Z2$ that behaves this way.

If there were zero upward drift and all step sizes were 1, it would take approximately $(|X|/2)^2 = 2,500$ generations on average for $Z1$ or $Z2$ to reach 50, thus removing $M2$ or $M1$ from the population. Larger step sizes and upward drift make this estimate an upper bound.

Population size has at least a quadratic effect. Were the population size $|X| = 10$, drift would occur in fewer than 25 generations on average.

The terms *premature convergence* and *drift* refer to the same final computational outcome but differ in their emphasis on its cause. The former suggests that the population converged too quickly, before adequately exploring the search space. The latter means that the population is too uniform, despite possibly having generated a variety of good solutions in past generations. I prefer the term *drift* because it is defined in terms of the present population X without reference to history, and because it steers one toward a remedy, called *diversity maintenance* (at the risk of apparent advocacy of political correctness). Methods to maintain diversity include increasing population size, perturbation with a heavy-tailed distribution, exploration, randomness, forbidding solution multiplicity in a population of discrete solutions, requiring a minimum distance between continuous solutions, and retaining a set of disparate incumbents. For more information about these methods, see Sastry et al. [105]. This active research area has received some serious attention by Dirk Sudholt and Carsten Witt (Sudholt and Witt [115]), Pietro Oliveto and Christine Zarges (Oliveto and Zarges [85]), and Tobias Friedrich (Friedrich et al. [38]), among others.

Unfortunately, the term *diversification* has become ambiguous. Some writers use it to mean *diversity maintenance* rather than *exploration*, as if the distinction between the latter two terms was not already sufficiently blurry.

The next sections present the basics of simulated annealing (SA), genetic algorithms (GA), and particle swarm optimization (PSO). To my mind, these are the most important NIAs. I will describe a variety of other NIAs later.

3. Simulated Annealing

Simulated annealing (SA), invented by Kirkpatrick et al. [66] and independently by Černý [19], is a general-purpose NIA. It relies on a black box to compute objective values $f(x)$ and a definition of neighborhood so that, given any solution x, the set $N(x)$ of the neighbors of x is readily produced. Unlike most NIAs, it does not use populations. Nonetheless, it is important and has been in wide use ever since the mid-1980s.

SA is inspired by the observation that minimizing the free energy of a physical system (e.g., an assemblage of molecules) is NP-hard in general yet appears to be accomplished in the physical world in a nonexorbitant amount of time. Let x denote a state or configuration of the system. For example, x_i could denote the orientation or spin of molecule i in a lattice. Let $H(x)$ denote the energy of state x. Annealing is a process of heating followed by gradual cooling, used by metallurgists and crystallographers to bring materials to low energy states so as to minimize internal stresses or grow large near-perfect crystals. Therefore, annealing minimizes the function $H(x)$. The idea of SA is to mathematically imitate annealing and apply it to functions $f(x)$ that we want to minimize, rather than state energy functions $H(x)$, which are of little relevance to OR problems.

The Metropolis algorithm, invented in 1953 by Metropolis et al. [77], mathematically simulates the transitions between states of a physical system at a given temperature. (For example, a transition could be a chemical reaction between two molecules, or the change of spin of a particle.) According to the Boltzmann distribution, the probability of being in a state with energy level α when the temperature is T is proportional to $e^{-\alpha/\mathrm{kT}}$, where k is the Boltzmann constant. For all x, let α_x denote the energy level of state x, and let $N(x)$ denote the set of states that are one transition from x. A single time step of the Metropolis algorithm, starting in state x, proceeds as follows.

Algorithm 3 (Metropolis Algorithm)

Step 1. Select at random a state $s \in N(x)$.

Step 2. Set $\Delta = \alpha_s - \alpha_x$.

Step 3. If $\Delta \le 0$, set $x = s$. If $\Delta > 0$, set $x = s$ with probability $e^{-\Delta/\mathrm{kT}}$.

Repeated application of the algorithm yields convergence to the Boltzmann distribution in the limit. Thus the algorithm simulates a physical system that has exponentially many states $s \in \mathcal{S}$, without calculating the sum $\sum_{s\in\mathcal{S}} e^{-\alpha_s/\mathrm{kT}}$. That is the principal benefit of the Metropolis algorithm.

As given, the algorithm yields the steady-state distribution at the given temperature T. It is instructive to interpret it in terms of neighborhood search. At $T = \infty$, it moves randomly in the search space. At $T = 0$, it performs local improvement—it only makes energy-decreasing (or energy-neutral) or moves. At intermediate temperatures, it makes both decreasing and increasing moves, favoring the former more at lower temperatures.

Simulated annealing starts with T very large and gradually reduces T to zero. This mimics the physical annealing process of heating material to a high temperature and gradually cooling it. To minimize $f(x)$ over $x \in S$, first define a neighborhood function $N: S \mapsto 2^S$ that gives the analogue of the states a single transition from a given state. We require $y \in N(x) \iff x \in N(y)$. Usually, $N(x)$ is defined to be the set of solutions within some fixed distance from x, excluding x itself. The most widely used version of the algorithm uses parameters *epoch length* $L \ge 1$, *cooling*

rate $c < 1$, and maximum and minimum temperatures $T_{\max} >> T_{\min} > 0$. Typical values of c are in the range $[0.01, 0.2]$, although values up to $c = 0.5$ are used occasionally. Typical values of L are at least of order n, the diameter of the search space with respect to $N()$. Because k and T appear as a product in the Metropolis algorithm, $k = 1$, with no loss in generality.

Algorithm 4 (SA Algorithm)

Step 1. Choose an arbitrary initial solution x. Set temperature $T = T_{\max}$.

Step 2. Do L iterations of the Metropolis algorithm with $\alpha_s \equiv f(s)$ at temperature T starting from x.

Step 3. If $T > T_{\min}$, reduce the temperature to $T = (1 - c)\,T$ and go to Step 2. Otherwise, set $T = 0$ and go to Step 3.

Step 4. Perform the Metropolis algorithm until no decrease in $f(x)$ has occurred in $L \log L$ consecutive iterations or local minimality is verified.

Note the following:

- Step 2 of SA is called an "epoch."
- In practice, it makes little difference for SA to store the incumbent because the solution at termination usually is the incumbent.
- As anyone who has implemented local search knows, Step 2 of the Metropolis algorithm can usually be sped up by a factor of order n by computing the difference $f(s) - f(x)$ rather than computing $f(s)$ from scratch.
- Step 3 of SA implements the *cooling* or *annealing schedule.*
- Step 4 (and Step 2 at low temperatures) can often be implemented more efficiently by searching $N(x)$ methodically.

An intuitive explanation for the success of the algorithm is that its partial selectivity at moderately high temperatures traverses many parts of the search space and finds a globally good region. As the temperature continues to decrease, the search gradually narrows to a small portion of the region wherein the algorithm finds a local minimum that is likely to be globally optimal or near optimal. This intuition is incomplete. The exponential form in Step 3 of the Metropolis algorithm, taken from the Boltzmann distribution, is anything but arbitrary. It is the unique form with the following property: for all neighbor-to-neighbor paths from x to z along which $f()$ does not decrease, the product of the transition probabilities has the same value (namely, $e^{-[f(z)-f(x)]/(kT)}$). This is the essential reason that the chance of escaping from a local minimum turns out to depend on how much $f()$ must increase en route to decreasing to a better solution (Hajek [50]).

In terms of exploration versus exploitation, SA begins doing almost pure exploration, gradually increases its emphasis on exploitation as the temperature drops, and ends doing pure exploitation at zero temperature.

Now that there has been considerable computational testing, SA rarely requires extensive tuning. The Boltzmann probability for $\alpha = 0$ is only 10% larger than for $\alpha = 0.1kT$, so all states with relatively low energy, $\alpha \leq 0.1kT$, have nearly equal probabilities. Therefore, to determine a "large" initial temperature, let $f_{\max}$ be the largest objective value of $\theta(n)$ randomly generated solutions, and set $T_{\max}$ in the range $2f_{\max}$ to $10f_{\max}$. At the other extreme, to determine a "small" terminating temperature, observe that at $\alpha = 5kT$, the e^{-5} term reduces the probability by a factor of more than 100. Hence all states with relatively high energy have very small probabilities. If μ is a lower bound on the optimum value, set $T_{\min} = \mu/5$. If no lower bound is available, terminate after there has been no improvement in the incumbent value for a small multiple of L iterations. SA should be able to traverse the entire search space during a single epoch. Therefore, initially set L to be on the order of $n\,|N(x)|$. The range 0.01–0.5 for the cooling rate c varies the run time by a factor of nearly 70, enough to turn hours into minutes. If that range is not enough to get to the maximum acceptable run time, adjust L.

4. The Basic Genetic Algorithm

Genetic algorithms are based on an analogy with evolution. Natural selection successfully produces organisms that are very well adapted to their environment. A mathematical simulation of natural selection ought to successfully produce solutions that are very well adapted to a mathematical environment that rewards good objective function values.

At a high level, a genetic algorithm maintains a population of N solutions. At each iteration, it selects a mating subset of the population and then builds a new population by recombining and mutating the mating subset. Here is the basic genetic algorithm (GA) for the discrete minimization problem, as popularized by Goldberg [47]. To emphasize the analogy to evolution, make the intuition clearer, and be consistent with GA literature, I cast it as a maximization algorithm.

Algorithm 5 (GA to Maximize a Nonnegative Function f)

Step 1. Randomly generate an initial population X of N solutions.

Step 2. For each $x \in X$ set $p(x) = \frac{f(x)}{\sum_{s \in X} f(s)}$.

Step 3. Initialize Z to be an empty population. Repeat $N/2$ times:

(3a) Select two (not necessarily distinct) members of X, x and y, each an independent sample from X according to the probability distribution p.

(3b) Perform recombination on the pair x, y to generate two solutions z, z'.

(3c) Add z and z' to Z.

Step 4. Perform mutation on the members of Z.

Step 5. If a termination criterion is met, stop.

Step 6. Set $X = Z$ and go to Step 2.

Note the following:

- Some implementations of Step 2 eliminate all but the M best solutions in X for some $M < N$ such as $M \approx \sqrt{N}$.
- See Section 2.5 for some alternatives to the selection rule embedded in Steps 2 and 3a.
- Some implementations of Step 4 use a second parameter, $\epsilon_0 > 0$. For each $z \in Z$, they perform perturbation on z with probability ϵ_0. Another common way to reduce the overall perturbation rate is to reduce ϵ.
- Typical termination criteria in Step 5 include upper bounds on CPU time, iterations, or function evaluations; zero or small improvement in the best objective value; and attainment of a desired solution quality.
- To preserve high-quality solutions in X that by randomness could be lost to selection, recombination, or perturbation, either maintain the incumbent[1] or modify Step 5 to the following: Step 5. Set $X = X \bigcup Z$. Reduce $|X|$ to N by retaining only the N elements with largest (best) function values.

Because X and Z are multisets, do not remove duplicates when taking their union.

It has long been known (see, e.g., the 1986 article by Grefenstette [49]) that GAs require extensive tuning. To be more precise, the performance of GAs varies greatly as a function of their parameter values. The parameters include the population size N; the number of crossover sites (see Section 2.4); offset and/or scaling values for f before computing $p(x)$ (see Section 2.5); the mutation and crossover probabilities, termination criteria, and possibly M, ϵ_0, and β. In 1994, Michalewicz [78] reported running more than a total of 1,000 trials on five transportation test problems (yes, transportation problems) to tune his GA, which was then run on cases up to size 10 by 10.[2]

GAs are also frequently tailored to a specific problem. A Google Scholar search on "new crossover" or "novel crossover" yields more than 7,000 results; a search on "new mutation operator" or "novel mutation operator" yields more than 2,000; and a search on "novel," "new," or "improved" selection operator yields more than 600.

Based on my limited sampling of the vast GA literature, perturbation is credited with furnishing exploration, and recombination is credited with furnishing exploitation (Goldberg [47], Yang and Karamanoglu [132]). I believe these correspondences are an oversimplification. Each should get credit for both.

Most perturbations change a solution slightly. Hence perturbation of a high-quality solution x exploits the region close to x. A sequence of small perturbations could arrive at a solution distant from x, but because selection is applied to every population generation, a sequence of worsening perturbations has a remote chance of occurring. On the other hand, perturbations that draw on heavy-tailed distributions contribute to exploration.

Recombination of a pair of distant solutions is apt to produce offspring close to neither one another nor their parents. For instance, suppose x and y are random binary strings of length n. Then $H(x, y)$ is distributed as a $\text{Bin}\left(\frac{1}{2}, n\right)$ distribution with mean $n/2$ and standard deviation $\sqrt{n}/2$. A crossover point j will produce z, z' with binomially distributed Hamming distances whose mean values are $E[H(x, z)] = (n-j)/2, E[H(z, z')] = n/2, E[H(y, z)] = j/2$ (and symmetrically for z'). Thus if all possible values of j are equally likely, the expected Hamming distance from x to one of its offspring is $n/4$—not as large as $E[H(x, y)]$ but quite substantial. Moreover, the crossover offspring are very unlikely to be in the convex hull of the previous generation. Therefore, recombination often provides exploration. On the other hand, if selection or drift has created a cluster of similar solutions, recombination within that cluster will exploit that cluster's terrain.

GA research continues to flourish. A Google Scholar search on "genetic algorithm" finds more than 11,300 publications for the 12-month period from June 2017 to May 2018, slightly more than linear and integer programming combined. One might wonder why. The basic GA algorithm is easy for an amateur programmer to code in 100–200 lines. Both pseudocode and code are freely available, too, requiring little work to apply to a specific problem. However, the same is true for the less popular SA. Ease of use, then, cannot be the only explanation for GA's popularity. I speculate that the evolutionary metaphor is catchier and also more convincing because the power of evolution is much more well known than the power of annealing.

Early widely cited publications on GAs, including Holland's [53], DeJong's [26], and Goldberg's [47], all emphasize the distinction between *genotype* and *phenotype*. In biology, genotype is the genetic information of an organism with respect to its species, but we can simply think of genotype as the DNA or chromosomes of an organism. Phenotype is the actual physical organism. In a GA, a *phenotype* is a *solution to a problem*; a *genotype* is an *encoding of the phenotype*. For example, for the TSP, a solution is a tour on a graph. That is a phenotype. A tour can be encoded as a binary string, where each bit corresponds to an edge in the graph. The bit is set to 1 if and only if the edge is part of the tour. The binary string is a genotype. One could choose to work with a different encoding—for instance, a permutation of the vertex labels. These researchers believed, correctly, that some encodings work better than others. Their phenotype/genotype distinction is profound and important. However, in my view, they missed the opportunity to employ it effectively. Holland and Goldberg thought that binary string encodings are usually the best. From an OR point of view, their different choices of genotype are different choices of models. Choosing a good encoding was to them like choosing a good model is to us.

Section 8.1 describes an old yet innovative type of encoding, whose power has started to become recognized.

5. Particle Swarm Optimization

PSO algorithms modify their population by attraction, varied with some randomization. The book by Eberhart et al. [31], although dated, is a basic reference. More recent overviews are by Banks et al. [4, 5], Kennedy [65], and Poli et al. [98]. PSO is a highly active research area. According to Google Scholar, there were 7,000 publications on PSO in the most recent 12-month period. I will present PSO in its natural and most often-used form as an algorithm for continuous optimization. It can, with modifications, be applied to discrete optimization (Kennedy and

Eberhart [64]). The underlying metaphor is that of animals searching as a group for a food source. The animals could be a pack of wolves, plague of locusts, flock of birds, school of fish, etc. The main characteristics of insect swarms, fish schools, and bird flocks are their mobility and the tendency of their members to stay near each other as they move. Each individual animal is idealized mathematically as a point moving in $\Re^n$.

I do not find this metaphor compelling for global optimization, because the biological marvel of these animal groups is not that they succeed in finding the best food source, nor that they find food quickly. The marvel is their spatial cohesiveness and coordinated movement without numerous collisions. Scientists have not figured out how schools of millions of fish behave almost as one. For example, there does not seem to be enough time, when those in the front of the school change course, for that information to reach the back of the school as quickly as it does (Partridge [87]).

For mathematical optimization, there is no need for multiple particles to discover the same solution. One discovery of a solution is just as good as many. You might expect PSO to waste computational effort with many particles swarming near each other, effort that would be better spent by smaller groups exploring more of the search space. This concern has some validity, as evidenced by various modifications to PSO to increase exploration. But as will become clear, PSO tends to achieve a fair amount of exploration.

Algorithm 6 (PSO to Minimize a Real-Valued Function on $\Re^n$)

(At each time step, the basic PSO algorithm maintains each particle's location x_i, velocity v_i, and incumbent location y_i, its best location so far.)

Step 1. Initialize a population of N particles. For each $i = 1, \ldots, N$: assign to particle i a random location $x_i \in \Re^n$; set the particle's incumbent location $y_i = x_i$; set the particle's velocity $v_i = 0$.

Step 2. Let $g = \arg\min_{1 \le i \le N} f(y_i)$, the index of the particle whose incumbent location has the best objective value.

Step 3. For all i, generate random values ϵ_i, δ_i. Update the velocity to $v_i = \theta v_i + \alpha\epsilon_i(y_g - x_i) + \beta\delta_i(y_i - x_i)$.

Step 4. For all i, update the location to $x_i = x_i + v_i$;[3] if $f(x_i) < f(y_i)$ update $y_i = x_i$.

Step 5. If a termination criterion has been met, stop. Otherwise, go to Step 2.

A few clarifications are in order.

- If there are box or linear inequality constraints on the location, enforce the constraints at initialization (Step 1) and stop any particle that would exit the feasible region in Step 4 at the constraint boundary.
- The random variables ϵ_i, δ_i are often given a U[0, 2] distribution.
- The parameters α and β are often given the same or approximately the same values. If $\alpha >> \beta$, the algorithm is likely to converge prematurely to a local optimum. Conversely, if $\beta >> \alpha$, the particles may fail to converge.
- In the original PSO, θ was absent from the velocity update equation in Step 3, equivalent to $\theta = 1$. Shi and Eberhart [110] introduced it in 1998.[4] Decreasing θ over time decreases exploration in favor of exploitation, as decreasing temperature does in SA. Imposing a decreasing upper bound on velocity is also effective (Eberhart and Shi [29]).

Exploration in PSO occurs initially because starting locations are independent and random. Later, the varied velocities of the particles, because of their different individual histories, also promote exploration.

6. History

This section surveys the history of NIA optimization, emphasizing its many links to OR, and introduces several more algorithms. The first NIA, it is generally agreed, was the neural net proposed in the 1940s by Turing [124] (see Copeland and Proudfoot [23]).

6.1. Early Evolutionary Algorithms

As best I know, the first NIA intended to solve what we in OR call optimization problems was a GA invented by Bremermann and implemented by Salaff (in Bremermann [14]), and tested on linear programming. It uses populations, perturbation, and crossover. Between 1963 and 1965, Rechenberg [99] and Schwefel [106, 107] invented the *evolutionary strategy* (ES) to optimize shapes for wind tunnel performance. Their ES employs selection (fitness) and perturbation, but not recombination. As such, it is very similar to local search with randomization. It is an NIA because it is explicitly motivated by evolution in biology. It is especially important because of its novel concept of search in a parameter space, a concept I will discuss in Section 8.1. Unfortunately, this concept did not catch on at the time, and when ES was used without it, it was dominated by local search algorithms.

The next NIA, the evolutionary programming algorithm (EA), invented in 1966 by Fogel, Owen, and Walsh (Fogel et al. [36, 37]), evolved computer programs by perturbation and selection, but not recombination. This NIA had little success at the time. From an OR perspective, this seems to have been because raw computing power then was inadequate for brute-force trial and error to perform well. However, it has been suggested that the artificial intelligence (AI) community rejected the work of Fogel et al. for what we politely call "philosophical" reasons. If true, it would help explain why that line of research was rather dormant until the early 1990s.

Several researchers, including Bremermann [14] and Fogel (Fogel et al. [36, 37]), independently added crossovers to ES. Read Fogel's son's scholarly book for the full "fossil record" of ES, GA, and EA [35]. In 1975, Holland published his book on GAs (Holland [53]), which he had been developing independently since the mid-1960s. Holland's main focus was not optimization, but in the same year, his PhD student DeJong completed his dissertation on the application of GAs to optimization (DeJong [26]). These are generally considered the foundational publications on GA. The first GA conference took place in 1985 (see Grefenstette [48]). But the catalyst for the explosion in GA research did not arrive until nearly a decade and a half after DeJong's dissertation, when Goldberg published his book on GA (Goldberg [47]). Goldberg intended his writing to be simple, clear, and intuitive enough for self-study by anyone who knows algebra and a little computer programming. He succeeded. The annual publication rate on genetic algorithms had doubled to 2,000 from 1984 to 1989. Five years later, in 1994, the annual publication rate had jumped fourfold to 8,000. Twenty years later, in 2009, the rate was over 150,000. I do not mean to imply that Goldberg's book was solely responsible for these increases, as it has only 85,000 citations according to Google Scholar.[5]

6.2. Local Search, Tabu Search, and Simulated Annealing

Meanwhile, operations research had produced many key concepts now widely employed by NIAs. Of course, for continuous variables, local search dates back at least to Newton. In 1965, Lin [69] compared 2-opting with 3-opting for the TSP. He found that both were sensitive to the initial solution and that the latter gave significantly better-quality solutions. These findings were the earliest published evidence I know of that demonstrated the benefit of multiple starts and the critical importance of neighborhood definition. Lin [71] also found that a subset of edges occurred in nearly all the 3-opt tours; he proposed fixing those "consensus" edges and limiting the search to edges needed to complete the tour. This is the same concept of attraction to frequently occurring values described in Section 2.4.

Lin and Kernighan subsequently invented extraordinarily effective heuristics for graph partitioning and for the TSP (Kernighan and Lin [65], Lin and Kernighan [70]). I will describe their profound innovation in terms of binary strings. Search from an initial solution $x = x^0$ along a sequence $x^1, x^2, \ldots$ of small neighborhood moves, keeping track of all elements that have been changed from 1 to 0 since starting at x^0. I think of those elements as being frozen. To get x^{i+1}, the next solution in the sequence, choose the best solution in the neighborhood of x^i that does not change a frozen element. Continue the sequence, regardless of how the objective

value compares with x^0, until the frozen elements prevent any further move. To complete this single iteration of the heuristic, set x^0 to the best x^i in the sequence and unfreeze all elements. Iterate until until no x^i improves on the current x^0. The heuristic is designed to move away from a local optimum (with respect to the small neighborhood) without being drawn back to it after a few steps. Instead, the search sequence within an iteration is forced to explore points far away from its start. To the best of my knowledge, NIA has made little direct use of this heuristic. However, it influenced the entire field of heuristics, including NIA, by demonstrating the power of *exploration* and *escaping local optima*. Moreover, it is essential to an historical understanding of NIA concepts because of its relationship to tabu search.

Glover invented tabu search and published the first paper on the subject in 1986 (Glover [42]). It is simplest to define tabu search in terms of the heuristics of Kernighan and Lin [65]. Instead of freezing elements until all movement is blocked, tabu search freezes an element for T moves for some constant T. The list of frozen elements is called the tabu list. The future trajectory of the algorithm depends on both the current solution and the tabu list. Hence, cycling (i.e., returning to the same solution and tabu list) is unlikely, and it may be profitable to run the algorithm a great many iterations provided that the incumbent is stored. In my view, this basic version of tabu search is a mere variant of the innovative Lin–Kernighan heuristic. However, Glover and his colleagues soon augmented tabu search with other concepts. By 1989, as described by Glover [43], they had developed most of the main concepts of NIAs except for populations and recombination (which Holland had already invented), albeit without the sexy biological metaphors and terminology. These concepts include the idea of a metaheuristic control level that repeatedly calls a lower-level search algorithm. They also include the crucial ideas of *diversification* and *intensification*. Moreover, tabu search recognized the tension and complementarity between the two (Glover [43, 44]). In the tabu search literature, these ideas emerge from analyzing the effects of differing types of memory—for example, the tabu list is short-term memory—and different levels of algorithm control. Consequently, as Glover and Laguna explain [45] (as quoted in Sörensen et al. [111]), unlike in the NIA literature, diversification and exploration are similar but not identical concepts, as are intensification and exploitation.

Kirkpatrick, Gellat, and Vecchi's invention of SA, published in 1983 (Kirkpatrick et al. [66]) (and independently invented by Černý [19], published in 1985), was the other major development in NIA prior to 1989. SA was the first general-purpose optimization heuristic to be widely adopted. It was easy to use and offered the prospect of convergence to a global optimum. The OR and computer science community soon began to study SA carefully. The theoretical analysis began with Lundy and Mees's [73] proof of convergence and culminated in determining cooling rates necessary and sufficient for global convergence (Anily and Federgruen [3], Hajek [50], [82]). The monumental computational study of SA by Johnson et al. [60, 61] remains the definitive empirical analysis to this day. SA was also successfully used in real-world applications (Bertsimas and Tsitsiklis [9], Eglese [32]). I recall in particular that Adam Rosenberg applied SA to find frequency assignments that increased the capacity of the AT&T Los Angeles cellular telephone system by 25% at zero cost, because I helped reduce the run time (on a DEC VAX) from a week to about a day (see Tovey [121]).

6.3. Ants and Other Particles

New algorithms started to appear in the 1990s. Ant colony optimization (ACO) for discrete optimization appeared in a 1991 paper by Dorigo et al. [28] and the 1992 PhD dissertation of Dorigo [27]. I will describe it in terms of binary string solutions for a maximization problem. Each binary component $i = 1, \ldots, n$ maintains a numerical value p_i analogous to the amount of pheromone deposited on it. ACO employs a parameter $0 < \alpha < 1$ to dissipate old pheromone. Each ant adds pheromone to each p_i for which its ith component equals 1, in an amount

proportional to its objective value. Then each ant constructs a new solution, setting the ith component to 1 with probability proportional to p_i.

Algorithm 7 (ACO to Maximize $f(x)$ over Binary Strings of Length n)

Step 1. Initialize $p_i = 0$ for all i and initial solutions x^k for ants $k = 1, \ldots, N$.

Step 2. For all i, update $p_i = \alpha p_i + \sum_{k=1}^{N} x^k f(x^k)$.

Step 3. (Construct new solutions) For each k, ant k constructs its new solution x^k component by component, setting $x_i^k = 1$ with probability proportional to p_i unless constraints dictate the value.

Step 4. Go to Step 2.

ACO was inspired by observations that ants find shortest paths from the nest to food sources by laying pheromone on the ground during their return trips to the nest. Ants en route to a food source follow a pheromone trail on the ground. If there are two trails from the nest, a departing ant will choose the trail with heavier pheromone layer with higher probability. Suppose ants make the round trip to the food source in one minute along trail A but in two minutes along trail B, and initially, the trails have equally heavy pheromone layers. Let 100 ants depart. In expectation, 50 will choose trail A. After one minute, they return, having augmented trail A's layer, but B's will not have been augmented. Hence more than half of the 50 will choose A again. After two minutes, more than 75 ants will have added to trail A's layer, but only 50 will have added to B's layer. This positive feedback loop eventually leads to all ants choosing the shorter route. The same kind of feedback applies to subpaths, leading the ants to find a shortest path, provided there are enough scouting ants to try out alternatives.

However, Dorigo applied ACO to the TSP, not the shortest-path problem (Dorigo [27], Dorigo et al. [28]). Each ant starts at some vertex in the graph and chooses an edge according to the p_i of edges incident on that vertex. It traverses the edge, arriving at a new vertex, and chooses its next edge in the same manner, but observing the constraint that it may not revisit a vertex (until the last step). The computational results were very poor until Dorigo added standard OR heuristics such as 2-opt or 3-opt between Steps 2 and 3. I find it disingenuous to present these TSP results as an ACO success. More recently, an NIA algorithm used the Lin–Kernighan heuristic internally but attributed the good computational results to the NIA (Rosendo and Pozo [103]). On the one hand, it is a good idea to be skeptical as to why an NIA has very good performance on a truly difficult problem until you have carefully read the description of the implementation. On the other hand, there are reasons to believe that it is a good idea to integrate NIAs with classical OR techniques.

PSO was introduced in a 1995 paper by Kennedy and Eberhart [30], a paragon of both exposition and scholarly attribution. I described PSO in Section 5. The differential evolution (DE) algorithm for continuous global optimization was developed between 1994 and 1996 by Storn and Price [114] (also see Ahlers et al. [2]). Its outer framework is similar to PSO and GA, so I will state just the core solution generation and selection procedure here. DE iteratively updates each member of a population X of points in $\Re^n$ as follows.

Algorithm 8 (DE Procedure to Update $x \in X$)

Step 1. Randomly choose three distinct other points t, u, v in X. Set $y = x$.

Step 2. For $i = 1, \ldots, n$, independently with probability p, alter y_i by the formula

$$y_i = t_i + W(u_i - v_i).$$

Step 3. If no component of y was altered during Step 2, choose $1 \le i \le n$ at random and alter y_i by the formula in Step 2.

Step 4. If $f(y) < f(x)$, replace x with y in the population X. Otherwise, discard y.

DE's unusual recombination procedure employs four rather than two solutions. The parameter value W is called the *differential weight*. It is often set to 2. Except for Koza's 1992

reincarnation of evolutionary programming as genetic programming (Koza [68]) and some related work by operations researchers, which I introduce later in Section 8.1, we have covered the major NIA developments through 1999.

6.4. The Metaphorical Explosion

The Cambrian explosion, circa 540000000–520000000 BCE, is the time period during which most animal phyla first appeared. I cannot help but be reminded of it when I consider the explosion of NIAs during the time period circa 2000–2018 CE, when it seems most animal phyla contributed to the appearance of at least one new algorithm. Next, I briefly describe several of these. We can hope the NIA explosion does not last 20 million years.

Geem et al. [41] introduced the *harmony* algorithm in 2001 based on the metaphor of jazz musicians jamming. As an amateur musician who has jammed with professionals a few times, I may assert that their algorithm in no way resembles what musicians do. (That does not mean it is a bad algorithm.) From an initially randomly generated population X, construct each component y_i of a new solution independently, as follows: With probability p, generate y_i randomly (from among the allowable values of the component); with complementary probability $1-p$, choose a solution $x \in X$ uniformly at random and set $y_i = x_i$. In the latter case, with probability q, perturb the value of y_i slightly. Once the entire vector y has been constructed, compute $f(y)$. If the worst solution in X is worse than y, replace it with y. Otherwise, discard y and repeat until some stopping criterion is met.

Pham et al. [92, 93] devised a *bee* algorithm in 2005 that is somewhat similar to PSO with randomization and attraction but no velocity. There are two classes of particles: scout bees and forager bees. Scout bees always move randomly in the search space. Periodically, the K best solutions—the particles whose present locations have the best objective values—are identified. The other particles are attracted to those best solutions and then do local searches.

Karaboga [62] invented an *artificial bee colony* (ABC) algorithm, similar to the bee algorithm. In ABC, there are three classes of particles: scout bees, employed bees, and onlooker bees. Scout bees move randomly, employed bees perform local search, and onlooker bees are attracted to a discovered high-quality solution and become employed. If local search does not achieve improvement after a number of steps, the employed bee becomes an onlooker.

Yang and Deb [131] invented *cuckoo search* in 2009 for continuous optimization. At a high level, it is a straightforward NIA algorithm that employs only the principles of perturbation and searching from good solutions. Unlike—and better than—most NIAs, it has very few parameters. Its method of perturbation is innovative. The high level repeatedly updates a population X of initially random solutions in $\Re^n$ as follows: First, choose an $x \in X$ at random, construct a perturbation z of x, and replace x or another random $y \in X$ by z if and only if z has a better objective value. Second, remove from X the $\alpha|X|$ solutions with worst objective values. Replace those solutions with perturbations of solutions randomly chosen from those that were not removed from X. There is an analogy between the removal of a fraction α of the solutions from X and the fate of cuckoo bird eggs. A fraction of the eggs cuckoo birds lay in other birds' nests are detected and pushed out by the other birds. The perturbation operation is to add independently to each component of x a number drawn from the heavy-tailed Lévy distribution (a generalization of the Cauchy distribution). The effect of the heavy tail is that with high probability, one or a few of the components will be changed by an order of magnitude more than the others. Along most dimensions, x will be perturbed slightly; along a few dimensions, x will make a big jump. This perturbation probabilistically defines a neighborhood that gives more exploration than standard local neighborhoods, but it is much more structured than random movement. It shows promise in computational studies (Richer and Blackwell [102]) and might turn out to have longevity in continuous optimization.

Havens et al. [52] invented *roach infestation optimization* (RIO), a variant of PSO, in 2008. The idea of RIO is to alternate between individual search and attraction. Initially, particles

(cockroaches) individually search for a solution (location) with good objective value, supposedly analogous to a dark location. They then are attracted to nearby particles for a period of time, after which they revert to a period individual search, followed by attraction, etc. Yang's *firefly* algorithm, which appeared the next year (Yang [129]), is a simple attraction algorithm for continuous maximization that starts from the usual randomized population. Its attraction rule is one-sided; particles only move toward particles that have better objective values. Each particle x identifies those nearby particles, $y^1, \ldots, y^k$, with larger (better) objective value. If $k = 0$, the particle x does not move. If $k \geq 1$, x chooses which y^j to be attracted to. The probability that x will choose y^i is proportional to $f(y^i)$ and inversely proportional to the distance between x and y^i, in imitation of the apparent brightness of light some distance away.

There are far too many NIAs to enumerate. Some of the more catchy ones are shuffled frog-leaping (Eusuff et al. [33]), krill herds (Gandomi and Alavi [39]) (which are misnamed, because krill swarm), invasive weed optimization (Mehrabian and Lucas [78]), sperm motility (Raouf and Hezam [97]), and intelligent water drops (Hosseini [56, 57]) (which must be an unnaturally inspired algorithm). Surveys that list far more NIAs and do a nice job of taxonomy include one by Binitha and Sathya [10] that defines 16 categories and one by Boussaïd et al. [13]. The Fister et al. 2013 survey lists more than 70 non-GA algorithms but makes some classification errors such as clouds being biological rather than physical and harmony being physical rather than human or other (Fister et al. [34]).

6.5. Close Analogies

To round off this section, I describe a few NIAs that depend on a closer-than-usual analogy between the biological phenomenon and the problem to be solved. As Nakrani and I have observed (Nakrani and Tovey [84]), biomimetic decision making has tended to perform better when based on a close analogy rather than a weak analogy, the swarm intelligence Enron disaster being a prime example of the latter.

Passino [88] invented the bacterial foraging algorithm (BFA) in 2002 for decentralized control of a fleet of drones or other robots. BFA can be thought of as a member of the PSO family. Particles are called "cells." Particles alternate between movement at a constant velocity and pausing to alter velocity, called "running" and "tumbling," respectively, by biologists who study the flagella-impelled movement of bacteria. Tumbling is triggered by detection of a decrease in nutrient concentration, which corresponds to the objective function to be maximized. Changes in velocity are based on attraction to high-quality particles, negative attraction to nearby particles, and a velocity in the direction of recent objective function improvement. This is called "chemotaxis" in analogy to bacterial sensing of nutrients and chemical signals from other bacteria. Selection and reproduction at good solutions are even simpler than perturbation—each above-average solution splits into two identical ones, in imitation of reproduction by the lengthening and splitting of well-nourished bacteria. Exploration is done in the usual way, generating random solutions, and is called "dispersal." Passino and Liu's foundational BFA work (see also Liu and Passino [71]) has several noteworthy qualities. First, they validate the biological analogy by showing that BFA accurately simulates *Myxococcus xanthus* bacterial movement. Second, their motivation, to control members of a physically distributed system, is a closer analogy to the biology than mere optimization of a purely mathematical function. Third, they are explicitly concerned with the problem of noise, thereby revealing a significant benefit of PSO: its ability to "climb a noisy gradient." I think this is an important concept for engineering applications, where sensor error and nonuniform or discrete physical dispersal render the objective function f "noisy." A single agent will have trouble detecting a noisy gradient. Multiple proximate agents that communicate with each other will average out the noise and detect the gradient quickly with very high probability.

Sunil Nakrani and I devised the *honey bee algorithm* to solve a specific optimization problem that Sunil had encountered while working at a web-hosting center (Nakrani and Tovey [83]). The problem was to dynamically allocate servers among different websites that the center was hosting, to maximize income from service-level agreements. The problem was not trivial because web traffic at different sites is unpredictable and heavy tailed, and because a server experiences significant downtime, on the order of minutes, to be switched from one site to another (this is for security reasons). I had studied honey bee colonies with biologist Tom Seeley and colleagues John Bartholdi and John VandeVate a dozen years previously (Bartholdi et al. [6], Seeley and Tovey [108]) and was amazed at how closely and deeply Sunil's problem resembled the honey bee colony's problem of allocating foragers among different flower patches. So instead of developing a conventional heuristic, we imitated the honey bee colony's allocation mechanism and got excellent results, increasing efficiency typically by 10%–20% on both real and simulated data. At least one major provider implemented and used the algorithm, which led to a Golden Goose Award in 2016. I want to emphasize that we never tuned the algorithm. We chose all parameter values based on scaling and dimensional reasoning, prior to writing any code. We did verify afterward that the algorithm performance was stable over a fairly large range of parameter values. We also tested and verified the accuracy of the deeper level of the analogy, between the patterns of degradation when the allocation is unbalanced (Nakrani and Tovey [84]). This NIA was different from most others described in this tutorial because it was not intended to be a general-purpose algorithm or metaheuristic. On the contrary, we only developed it because of the surprisingly close homology between forager and web host server allocation.

Ant robotics emerged as a field in the 1990s, initially focused on building limited but cheap and easily programmable robots that could follow a path. By 2000, the field's goals had broadened to decentralized control of group tasks such as terrain coverage, accomplished by Koenig, Szymanski, and their students (Koenig et al. [69], Svennebring and Koenig [117]). Cicirello and Smith [20] invented ant colony control (AC^2) for robots on factory shop floors to route themselves autonomously, using only local stigmergic information in the form of artificial pheromone trails. (Coordination by placing and detecting physical cues in the local environment is called *stigmergy* in biology.) They found that AC^2 led to near-optimal routing even on complex layouts. In both of these cases, the problem to be solved (terrain coverage, task division, etc.) is quite similar to a problem ant colonies solve, and moreover, the constraints on an agent's capabilities are similar to an ant's perceptual and cognitive limits.

7. Theory, Computation, Applications, and Critiques

The apparent purpose of this section is to describe some of the theoretical studies, computational properties, and applications of NIAs. The less obvious but more important purpose is to critique the insularity and technical level that prevails in much of the NIA field, as well as to critique our own insularity and abdication.

Despite my forthcoming criticisms of the NIA field, I am convinced that some of its innovations are valuable and ought to be studied and incorporated into the OR algorithm toolbox. The crossover operation is one example. Using parallel processing for populations, and using populations to fill out a Pareto frontier for multiobjective problems, are two more examples. Later, in Section 8, I will describe what I think is the most important innovation.

7.1. Theoretical Analysis and Critique of NIAs

Virtually all NIAs for discrete optimization, for some parameter values, obtain a global optimum with probability 1 as the number of iterations goes to infinity, as long as they maintain an incumbent. For many NIAs this is simply because the set of states the NIA can be in forms a finite ergodic Markov chain. Ignoring the incumbent, the states are the possible populations

X; randomized solution generation and selection render the set of states a single communicating class. The reason is similar for the rest of the NIAs. Strict selection rules force some states to be transient, but for every state x, there is a state y that contains a global optimum and can be reached from x. For instance, the harmony algorithm randomly generates an $(N+1)$st solution from its population of N solutions, and of those $N+1$ solutions, it removes the one with worst objective value. Unless the newly generated solution is the one discarded, the algorithm can never return to its previous state. But it is obvious that the algorithm will almost surely attain a global optimum.

For continuous variables, the situation is murkier. At the very least, the algorithm must guarantee convergence to a local optimum, which is not computationally trivial in nonlinear programming. NIA researchers do not take such convergence for granted! As a case in point, Van den Berghand and Engelbrecht [126] prove that the original PSO can fail to converge to a local optimum. In general, NIAs for continuous variables are routinely tested on both unimodal (having one local optimum) and multimodal (having many) functions. Of the 140 test problems collected by Jamil et al. [59], 24 are unimodal functions. Of the 20 benchmark problems assembled by Tang et al. [118] for a large-scale global optimization competition, 8 are unimodal. I have never understood why these algorithms do not always include something similar to my Step 3.5 in Section 2.3.

NIAs with populations are technically difficult to analyze precisely, because the solutions interact with each other. It also does not help that the state space has cardinality almost 2^{nN}, where n is the dimension of the search space and N is the population size. And that is ignoring ancillary data. Most theoretical analyses focus on the trajectory of a single population member. Trelea [122] proves convergence of a deterministic PSO trajectory in one dimension. He argues that this proves convergence in multiple dimensions, but I think his argument implicitly assumes that the objective function is separable. Sun and Liu [116] give a proof of almost sure convergence of a single particle for a PSO under assumptions of continuity in a compact search space. However, their proof seems to assume convexity of level sets in a neighborhood of the optimum. Stability of some simplified PSO-type algorithms, in the sense of collision avoidance and convergent velocities, has been shown by Liu et al. [72] and Gazi and Passino [40].

These are weak guarantees at best. It is simple to construct examples for which the expected number of function evaluations far exceeds the cardinality of the search space.

Example 4 (Simulated Annealing). Run simulated annealing on the space $\{0,1\}^2$ with $f(0,0)=0, f(1,0)=f(0,1)=10^6$, $f(1,1)=1$. Starting at $(1,1)$, at temperature 10^5 or less, the expected number of iterations until the first arrival at $(0,0)$ is rather larger than 4.

Example 5 (Genetic Algorithm). Model the problem

$$\max\ y \text{ subject to } 0 \le y \le 32;\ \ y \text{ integer}$$

for solution by a GA. Let the search space be $\mathcal{S}=\{0,1\}^6$. Treating $x \in \mathcal{S}$ as a binary number, let $f(x)=x$ if $x \le 32$, and let $f(x)=0$ if $x>32$. Run the GA in Section 4 with population size $N=6$ and mutation probability $\epsilon=0.01$. The probability is $(63/64)^6 > 0.9$ that the initial population X does not contain 32. Conditioned on that event, the probability is $1-(32/63)^6 > 0.98$ that X contains at least one x such that $f(x)>0$. With probability of more than 0.88, therefore, Step 3 will generate Z consisting entirely of solutions in the range 1–31. For each $z \in Z$, the probability is less than 10^{-4} that Step 4 will mutate z to 32. Mutations to any z with $f(z)=0$ have zero fitness and may be ignored. Hence, if X has the property that $1 \le f(x) \le 31$ for all $x \in X$, the next generation has the same property with probability greater than $1-6(10^{-4})$. This proves that the expected number of generations until the optimum value $z=32$ is first attained is at least $(0.88)10^4/6$, corresponding to at least 8,800 function evaluations, compared with search space size $|\mathcal{S}|=64$. (Note that 8,800 is an underestimate because selection favors solutions with more than one bit set to 1, for which the probability of mutation to 32 is less than 10^{-6}.)

I chose the preceding example for its delicious irony. Except for its upper bound of 32 instead of 31, it is the unimodal "all 1s" discrete optimization problem popularized by Goldberg [47] to promote GAs and still used as a test case for NIAs.

Simulated annealing is the only NIA for which tight necessary and sufficient conditions for convergence are known. Lundy and Meese [73] were the first to prove convergence; Hajek [50] and Anily and Federgruen [3] found tight necessary and sufficient conditions using the theory of large deviations and Markov chain convergence, respectively. Other proofs were obtained by Mitra et al. [82] and Tsitsiklis [123]. The cooling schedule decreases T at a logarithmic rate, with a constant that equals the maximum over all points x in the search space of Δ_x. Note that Δ_x is the minimum, over all paths P from x to any point y with $f(y) < f(x)$, of the maximum of $f(z) - f(x)$ over all points z in the path P. From a geometric point of view, if you visualize the function f as giving the height of each point in the search space—this visualization is called the *landscape*—this constant is the least amount you must go higher than x to reach a point y lower than x.

The history of GA analysis reveals the mathematical naiveté characteristic of much of NIA research. Holland [53] formulated the idea of a *schema*, which for a vector of binary variables is a projection of the unit hypercube vertices onto a subset of its dimensions (i.e., the vertices of a face of the hypercube). (More generally, a schema can be the intersection of a set of hyperplanes with the search space. Here, I restrict discussion to binary variables.) He conjectured that regions of high-quality solutions would be defined by schema. If one could search simultaneously through different schemata, one could optimize efficiently.

Holland made a claim, called the schema theorem [53], which suggested GAs would heavily favor good schemata over poor ones. He analyzed the probability that the ith component of a random member of a GA's population would equal 1 and gave a now-disputed proof[6] that it is close to the probability of choosing the historically better lever in an optimal strategy for a two-armed bandit problem. The connection to the bandit problem is important. In the terms of this tutorial, the optimal solution to a bandit problem is an optimal trade-off between exploration and exploitation. (In multiarmed bandit problems, exploitation corresponds to pulling levers that historically have the best expected payoff per pull; exploration corresponds to pulling other levers that might have better expected payoffs per pull but are not known as such because, for instance, their payoffs have high variance.) Largely on these bases, Holland argued that GAs would globally optimize efficiently, despite the exponentially many schemata.

Besides appealing to schemata, Goldberg [47] advocated the "building block hypothesis" that predicted GAs would optimize efficiently. A *building block* is a substring in a specific solution location that makes solutions have good objective value. For example, $x_{20} \circ x_{21} \circ x_{22} \circ x_{23} \circ x_{24} = 11{,}100$ would be a building block if changing the 20th–24th components of a random solution y to 11,100 usually improved y's objective value. If building blocks exist, crossover would be apt to generate solutions containing multiple building blocks, thereby optimizing quickly. However, analysis and empirical studies by Mitchell et al. [80, 81], O'Reilly and Oppacher [86], and Burjorjee [15] have definitely not supported the hypothesis that building blocks exist for nonunimodal problems (see also McCall [75] for a discussion).

In 1997, Wolpert and MacReady published a theorem called the "no free lunch theorem" (NFLT) that got a lot of attention from the GA and the rest of the NIA community (Wolpert and MacReady [127]). When I first saw this theorem, I could not understand why it had been published, much less raised such a fuss. The theorem is mathematically obvious and trivial. It says that if f is an arbitrary real-valued function on a finite set S, then all search strategies to minimize $f(x)$ over $x \in S$ are equally effective, in terms of the number of evaluations of the function f. The proof is immediate by symmetry among all permutations of the set S, because f is completely arbitrary. In other words, all a search algorithm can do is evaluate $f(x)$ for one x at a time. Each evaluation provides no information about the values of f at other points in S. Therefore the choice of the next x to evaluate is irrelevant to the algorithm performance.

Why then, as Yang writes [130], did the NFLT "[send] out a shock wave" (p. 18) through the NIA research community? I think it is because they were mathematically ignorant. NIA research is permeated with two mutually contradictory beliefs: the first is that their metaheuristics work well on any problem; the second is that their algorithms do not make any use of problem structure.[7] NFLT showed the contradiction between these two beliefs. For a fuller account of GA analysis, see Reeves's review of GA [101].

To summarize, the inventions of NIAs by Bremermann, Fogel, Holland, Kirkpatrick, Dorigo, Kennedy, and their colleagues were, on the whole, exceptionally creative but mathematically unexceptional. They seemed unaware of simple discrete examples for which their algorithms were slower than complete enumeration and of continuous examples for which there was very slow or no convergence. They also mistook the hill-climbing aspect of their algorithms for the genius of the natural world, as though the neighborhoods defined by their encodings had nothing to do with the hill climbing's success. That is why the NFLT surprised them. On the other hand, the scarceness of exact analyses of population-based algorithms ought not to be criticized, because such results are technically very difficult to achieve.

7.2. Computational Performance, NIA's Insularity, and OR's Failure to Engage

The extensive study of SA by Johnson et al. [60, 61] revealed several striking properties of the algorithm. A single long run of SA outperformed multiple shorter independent runs. SA had a wider time/quality trade-off range than other heuristics, which tended to level off. That is, given enough time, SA in most cases outperformed other heuristics being given the same amount of time. The logarithmic cooling schedules prescribed by theory were much too slow to be used for computation.

I have avoided reporting computational results for each specific algorithm, because they can be summarized easily. Unless they are specialized and tuned, most variants of GA and PSO are considerably slower than SA, which had previously held the record for slowness.

Computational studies of continuous optimization NIAs have been largely unpersuasive and yielded inconsistent conclusions, except for a broad consensus that today's NIAs do not scale up well. Test problems are almost invariably small to medium sized, with 2–40 variables. For example, Hassan et al. [51] use a test suite of 8 problems, 7 of which have fewer than 10 variables. Civicioglu and Besdok [21] compare PSO, cuckoo search, ABC, and DE on a suite of more than 50 problems whose median and maximum numbers of variables are 5 and 30, respectively; roughly half of the 100 test functions in a benchmark set (Yang [130], appendix A) have 2 variables. During my preparatory reading for this tutorial, I noticed that one of the two test problems in a comparative study solves to optimality immediately at the LP relaxation (Saka et al. [104], p. 41) when modeled as an integer program.

NIA solutions to discrete optimization problems vary considerably in quality. Among them there certainly are many impressive results. However, it is usually unclear to what the success should be attributed. It could be a specialized mutation or crossover operator, developed after much trial and error; it could be expensive tuning on a large set of parameters. It could be the use of an OR technique as a subroutine; such NIAs are called *hybrids*, a term that can overemphasize the power of the nature-inspired part of the algorithm. In Section 6.3, I gave an example of an NIA that made repeated use of the Lin–Kernighan TSP heuristic. I have seen only a small subset of the high-quality NIA-produced TSP solutions on medium to large instances, but all in that subset used local search or SA.

Some of these studies ineptly report CPU times or iteration counts rather than the numbers of function evaluations or other operations. OR has had standards for reporting computational results since the 1970s (see Crowder et al. [24]; also see the Barr et al. guidelines [6] and Hooker's [55] critique of competition as opposed to experimentation). However, my principal criticism of the NIA computational studies is their insularity. In the NIA literature, it is the

rule, not the exception, to make comparisons only with other metaheuristics, and usually only with other NIAs.

Goldberg's PhD dissertation research is a prototypical example of insularity. His GA found close-to-optimal solutions to a pipeline problem whose optimal solution had been found 15 years previously by Wong and Larson via dynamic programming (Goldberg [46], Wong and Larson [128]). It was due only to Wong and Larson's results that Goldberg knew his solutions were near optimal. In traditional OR, an algorithm that finds suboptimal solutions in orders of magnitude more time than it takes to find optimal solutions would not be publishable, much less a research breakthrough. Yet Goldberg writes that on "pipelines . . .genetic algorithms have been used successfully" (Goldberg [47], p. 136).

In a 1992 *Scientific American* article, Holland [54] wrote glowingly about the pipeline GA research, with nary a mention of Wong and Larson's [128] method, which on contemporary computers would have found optimal solutions in seconds. Holland was Goldberg's PhD advisor, so surely he knew of Wong and Larson's work, but he deemed it irrelevant. I am confident Holland acted with full propriety[8] *according to the standards of his academic community*. My trouble is that I do not grasp what those intellectual or scholarly standards are. And I expect that most of my OR colleagues don't, either.

The gap between GA and dynamic programming did not narrow. In 1998, Carter solved a more complicated type of pipeline operation problem via what he called "nonsequential dynamic programming" (Carter [16], p. 7). His algorithm solved his toughest test case in 1.5 minutes, bringing fuel consumption from 91.62 to 63.8 (IBID, p. 15). It is worth quoting his comparison with GA: "The genetic algorithm . . . reduced fuel consumption . . . to 71.36. . . . Counting the time spent tuning the method. . . roughly one CPU year has been spent on the genetic algorithm getting a highly suboptimal solution—over 300,000 times slower than nonsequential dynamic programming" (IBID, p. 17).

ACO is based on the method described in Section 6.3 by which ants find a shortest path on a graph with positive edge lengths, a method thousands or more times slower than the OR state-of-the-art algorithms from 50 years ago. The number of papers that test an NIA on the TSP, without regard to the contemporary TSP software that provided the optimal tour lengths, is staggering (as evidenced by Google Scholar results; see also Bonabeau et al. [11] and Michalewicz [78]).

I compared NIA performance on the TSP with Cook's Concorde results (Cook [22]) in a talk at the American Association for the Advancement of Science (AAAS) annual meeting early in 2018. (AAAS publishes the premier journal *Science*). It upset me that the attendees—mainly top-notch scientists—were very surprised to see the factors of 100 to 1,000 difference between the number of cities in an instance solved optimally. And they did not know enough to appreciate the difference between finding an optimal solution and proving optimality. I think this is a failure of the OR community.

Moreover, it is just one example of our overall failure to deal with NIAs. We may think NIAs are beneath us, but they threaten to bury us. Instead, we should address them in the same way we addressed simulated annealing: with careful theoretical and computational study. We should not let absurd claims of efficiency or optimality in the general science and popular press go unrefuted. We should challenge the NIA field's insularity, whatever its cause. I assert that there is neither a mathematical nor an engineering justification for comparing NIAs with only other NIAs.

If NIAs are general-purpose algorithms, their generality must be tested, and they should be compared with other general-purpose algorithms. But how general are NIAs? "Probably everybody's GA is unique!" exclaims Reeves in the conclusion of his chapter on GAs (Reeves [100], p. 75). If the amount of parameter tuning and specialization of generation and selection operators renders an NIA a specialized algorithm, it should be compared with other specialized algorithms.

I call attention to a recent paper by Sergeyev et al. [109] in the *Scientific Reports* mega-journal. In a purposeful effort to bridge the gap between the NIA and math programming communities, it compares the performance of both types of algorithms on 800 randomly generated global optimization instances. The authors seem unaware that generating instances

with known optima can only produce NP-complete rather than D^P-complete test cases (see Rardin et al. [98]). Nonetheless, rather than sniping at this study because all of its instances have five variables and known optima, we should ask ourselves why we have not been conducting studies like this.

To summarize, NIAs would generally compare poorly with OR algorithms in both speed and solution quality, were the NIA community to make such comparisons. NIAs that do perform well on discrete problems are apt to be those that have incorporated OR methods internally. NIAs have not scaled up well on continuous problems. We in OR have, for the most part, ignored NIAs; we have failed to perform the rigorous comparative testing and experimentation that the NIA community does not do. Even in the few cases such as the TSP and convex or unimodal optimization for which we have strong evidence of the superiority of OR methods, we have failed to inform those in the sciences, engineering, and industry.

The OR community ought to engage with the NIA community. From my experience in interdisciplinary research, successful engagement will require time and effort. The words "solve," "intractable," "efficient," "proved," etc., do not mean the same thing to us as they mean to them. Inconsistent vocabularies suffice to break an engagement. For starters, I would send several OR optimizers to a couple of NIA conferences. After they gain a sense of that culture, they would educate the rest of us at OR conferences.

7.3. NIA Applications and OR's Territorial Abandonment

When I began work on this tutorial, I thought that NIA insularity was an intellectual laziness, a scholarly sloppiness, feebly excused by the formula, "It's not fair to compare general-purpose algorithms with specialized ones." I was profoundly mistaken. I now think there has been a progression through four types of insularity.

The Four Stages of NIA Insularity

1. *The Solipsist*: Feels the world exists insofar that it supports his algorithm; uses what helps with no thought of comparison (e.g., Goldberg's pipeline research).

2. *The Separatist*: Cognizant of OR; believes it to be a separate domain. For example, "It is common to use metaheuristic algorithms in solving nondifferentiable nonlinear-objective functions the solution of which is either impossible or extremely difficult by using the classical optimization techniques" (Civicioglu and Besdok [21], p. 316).

3. *The Secluded*: Not cognizant of OR optimization. Took courses, reads books and journals, attends conferences, and uses software, all different from OR. For example, the NFLT "sent out a shock wave to the optimization community" (Yang [130], pp. 18–19). Because it did not even cause a tremor in OR, I infer that the writer is oblivious to the OR optimization community's existence.

4. *The Supremacist*: Cognizant of OR optimization; believes it to be superseded by NIAs. For example, Jamil and Zepernick wrote the following in 2013:

> Optimization methods, once dominated by the classical approaches, are constantly encroached by MAs [metaheuristic algorithms] due to their (i) effectiveness, (ii) reasonable computational time, (iii) broad flexibility, (iv) intuitive mathematical formulation, and (v) ability to handle uncertain, stochastic, and dynamic information. . . .These methods have been proven to be effective in handling problems that were previously thought to be computationally too complex. (Jamil and Zepernick [58], p. 50)

I believe that the solipsist was the predominant type in the early years of NIAs; I fear that the supremacist is becoming the predominant type. How could this be occurring? I posit four stages, corresponding to the four types of insularity. First, OR vacated part of optimization territory while NIAs were coming into existence. Second, NIAs moved into that territory.[9] Third, NIAs spread rapidly throughout their territory because they were easy to use. Fourth, NIA has spread into newly created territory and is encroaching on OR's territory.

I will elaborate. First, let us identify the territory that OR vacated.

I just queried Google Scholar with the words "genetic algorithm application." The first applications I saw included bankruptcy prediction, classifying breast tissue as mass or normal, electromagnetic optimization, calibrating conceptual rainfall-runoff models, flow shop problems, fashion design, fuzzy control, designing neural networks, and crystal structure solution from powder diffraction data. My query about PSO applications returned, in the first two pages, unit commitment for power systems, text clustering, classifier for determining candidates for thoracic surgery, neural network to analyze outcomes of construction claims, skull prosthesis modelling, estimations of oil demand in Iran, assembly scheduling of a two-stage distributed database, estimations of parameters in nonlinear engineering systems, optimal location of flexible AC transmission systems, recognition of control chart patterns, and solutions to geophysical inverse problems.

Poli [94] analyzed the 700 publications on PSO applications in the IEEE Xplore database as of 2007. I estimate that a similar analysis now in 2018 would encounter six times as many publications. He divided his 700 articles into 26 categories. These included antenna design, biomedical (ranging from tremor analysis to phylogenetic tree reconstruction to drug design and radiotherapy planning), electric power distribution, control, neural networks, design, neurofuzzy systems, combinatorial optimization, image analysis, robotics, graphics, electronics, motors, etc.

Now compare the range of the applications listed here with the range of real and claimed potential applications in OR and math programming journals. Except for electric power applications, you will find few that deal with physical designs or phenomena such as chemical processes, chemical structure, electronics devices, prosthetics, or antennas. You will find few if any applications to image analysis, control, neural net training, and few exotic ones such as fashion design or calibrating rain runoff models.

From a mathematical point of view, you will find few continuous variable nonconvex nonlinear problems, few real-time control problems, and few problems for which the objective is not quantified or constraints are fuzzy. Those of us in academia know perfectly well why such applications are rare. They do not lend themselves to nice rigorous mathematical results. As OR has become more mathematical, they have become less appealing to our taste and less rewarding in academic currency. The consequence is that, as my colleague Ted Pavlic says,

> Although nature-inspired metaheuristics have questionable demonstrations of performance, they fill a mathematically inconvenient void of nonlinear and nonconvex optimization problems that continues to be avoided by the traditional OR community despite its relevance to realistic applications. (Pavlic [89])

That is the territory most OR researchers abandoned.

In stage 2, NIA researchers worked on many different kinds of problems, including some firmly in OR's domain (e.g., the TSP), but they often purposely steered toward ill-defined, stochastic, nonlinear, and otherwise mathematically ugly problems, which (as separatists) they knew to be in their domain.

In stage 3, the NIA community grew rapidly in numbers of annual publications, active researchers, topical journals, and conferences. Insularity aided growth by removing the barrier of OR research standards, and catchy metaphors made the NIAs appealing. But the major spur to growth was the simplicity of the NIAs themselves. Practitioners and novice researchers alike could easily and rapidly get their code running and, if need be, linked to a separate function evaluation module.

In my forecast of stage 4, generality and ease of use continue to impel the spread of NIAs. Neural networks come back into vogue, and presto—neuro-evolutionary algorithms appear to train them. In the meantime, OR has not made its tools much more general or easy to use. NIAs gain ground because they offer a smaller but more rapid return on a smaller investment than

OR. To quote Pavlic, "People use nature-inspired metaheuristics not because they want more OR; they do it because they want LESS" (Pavlic [89]).

According to Poli [94], practitioners want something simple, reliable, quick to learn how to use, and usable as a building block with other tools they already know.

One possible future would be that metaheuristics become a standard way to take the first crack at an optimization problem—a computer-powered quick and dirty method. If the solution quality within time constraints is not good enough, or the potential benefit from a higher-quality solution is large, try a more sophisticated or tailored method.

I do not know at all what the outcome will be. But I am certain that we in the OR community should not just let things happen to us. We should face the situation squarely and take conscious deliberate action. I am also certain that we should not permit an outcome wherein the NIA community has demonstrably false beliefs about the supremacy of their methods.

8. Opportunities

8.1. Unconventional NIAs

I think the most promising avenue of NIA research is to employ the analogy to biology, that an organism's DNA is a formula for creating the phenotype. Your DNA is not a microscopic homunculus; it is an algorithm for constructing your body. By altering just one DNA site, scientists can make a fly develop a leg instead of an antenna. A small change to the fly's phenotype would probably be useless, or worse, not viable. That is the true genius of evolution—that change occurs in the genotype whereas selection occurs to the phenotype.

Many operations researchers have encountered the analog of this phenomenon when trying to use local search. In many problems such as integer programming, scheduling, or multivehicle routing, changing a bit in a solution string is extremely likely to produce an infeasible solution. Flip the value of one binary variable, change the start time of one job, or add one point to a vehicle's route. A bunch of other variables may have to be changed in a particular way to regain feasibility. The landscape of the problem has deep crevices around feasible solutions, with respect to hill climbing with a small Hamming distance neighborhood. Local search in the space of solutions can be futile.

Therefore, perhaps we should try searching through the space of computer programs, programs that produce solutions. This is what genetic programming attempts (Koza [68]). But that is too extreme an approach. It does not use information we have about the problem at hand. Instead, we should work with a known problem-specific heuristic controlled by a set of parameters (including instance data). *Search in the space of those parameters.* As long as the heuristic is designed to get a feasible solution, changing parameter values will not change the solution from feasible to infeasible. The landscape will have no deep crevices. Storer and his student Renzo Vaccari, who completed his thesis in 1990 (Vaccari [125]), did the first research of this type, to my knowledge. They got excellent results with GAs on jobshop scheduling problems, consistently bettering the best-known heuristic (Storer et al. [112, 113]).

Here is their setup. Fix the size of a problem instance, for example, n equals the number of points in a TSP. Let $\mathbb{I}$ be the set of instances. Let H be a heuristic controlled by a string of parameter values $s_1, \ldots, s_m$. For example, supposing m divides n, let the allowed values of s_i be 0 and 1. Given instance $I \in \mathbb{I}$, the heuristic H constructs a tour as follows: if $s_1 = 0$ (respectively, 1), choose the vertices of the shortest (respectively, longest) edge to be the initial tour T. Thereafter, for $i = 1$ to m, H adds edges to T until $|T| = ni/m$ according to the nearest (farthest) insertion rule if $s_i = 0$ ($s_i = 1$). That is, H uses either the nearest or farthest insertion rule to add the next n/m edges, depending on the value of s_i.

Let the pair (s, I) denote the tour that H builds on the instance I, guided by parameter string s. Therefore the set of all pairs (s, I) is the set of all possible tours. For arbitrary instance $\hat{I} \in \mathbb{I}$, define $L(\hat{I}, s, I)$ to be the length of the tour (s, I) in the instance $\hat{I}$. Then $L(\hat{I}, s, I)$ is the objective function to be minimized.

Searching in "heuristic space" means fixing $I = \hat{I}$ and running GA or another metaheuristic on the space of possible strings s. Searching in "problem space" means fixing s, initializing $I = \hat{I}$, and running a metaheuristic on the space $\mathbb{I}$ of possible I. One must be careful searching in problem space because the cardinality is huge, and straying far from the original point $I = \hat{I}$ would have little to do with the problem. Storer et al. [112, 113] limit the straying by requiring a large number of restarts from the same I before moving from the incumbent.

It is essential to define s, $\mathbb{I}$, and the form of the solutions (s, I) so that (s, I) is sure to be a feasible solution with respect to $\hat{I}$. For example, for a single-machine scheduling problem, it would be unwise to define a schedule (s, I) as a list of job start times if job run times depend on the data in I. Instead, define a solution (s, I) to be the order in which jobs are run.

There are hints of this idea in the conventional NIA literature (i.e., exclusive of genetic programming). For instance, in Dorigo's [27] application of ACO to the TSP (see Section 6), ants follow a randomized nearest neighbor heuristic to build their tours. Because the graph is complete, they are guaranteed to build feasible solutions. The algorithm changes the edge weights from iteration to iteration. This is an example of Storer et al. searching in problem space. The heuristic is the same greedy algorithm each iteration. The edge weights, which are the problem data, change each iteration. Theory to explain the success of Storer et al. is lacking, even though much the same idea has been used in AI for more than two decades (Michalewicz [78]).

The other really promising avenue I see is, again, inspired by looking more deeply into the biology, seeing something powerful, and imitating it in our algorithms. It is an old idea by now to use a metaheuristic to tune another metaheuristic. Inspired by the way some genes regulate other genes, and cells can regulate the genes in their DNA by producing proteins, the idea of self-tuning by self-regulation has been explored for a long time. An evolutionary metaheuristic can change its own parameter values and which mutation or crossover operators it uses. I think that most of the potential of this kind of control, especially via hierarchies of control genes, is untapped. For example, some evolutionary NIAs do not converge well to local optima because that would require too much intensification. Earlier in this tutorial, I proposed adding an explicit local search step. But why can't the NIA self-regulate to do intensification until zero or tiny changes in function value indicate local optimality, at which time the NIA could turn off an intensification gene and turn on an exploration one?

I imagine that in another decade or two, there will be evolutionary NIAs so large that no one fully understands what they do, that can turn long chunks of their genetic material on or off with a master control gene, whose chunks do the same within themselves, or even among themselves. There is no requirement that the program hierarchy be a tree. If these NIAs are the product of billions of generations of simulated evolution, they could be better optimizers than any program a human could write explicitly, just as no one knows how to write the champion GO program explicitly. All we can do is write the program that learns to play GO and run it for a long time.

8.2. Research Questions

In this section I state some questions that OR researchers may find interesting and be well equipped to tackle. I should caution that I have not scoured the literature to be sure all of these are unsolved.

1. The metaphor of diversity maintenance suggests that an NIA could monitor the population and intervene when appropriate. However, optimal drift detection is obviously NP-hard by reduction from max clique. (Represent edge (i, j) in the graph by a string of all 1s except a 0 in positions i and j. In a clique of size k, the edge vectors agree on all but those k positions.) Find fast approximate algorithms for detecting low diversity. Find fast algorithms to increase diversity.

2. Does increasing the population by a factor α always increase the expected time until diversity is lost by $\Omega(\alpha^2)$, assuming an initial 50–50 distribution between the two alleles? (See Example 3.)

3. How many virtual genders are optimal? For physical organisms, the difficulty of a triad rather than a dyad coordinating seems likely to outweigh any advantage conferred by the increased variety of offspring. Absent the physical disadvantage, why couldn't three or more genders be a superior means of reproduction?

4. Consider the space-filling curve heuristic for the Euclidean two-dimensional TSP. Run the Storer et al. [112, 113] heuristic in problem space, where a single change to the problem data consists of moving one city. This moves the city to a different place in the sorted order. Hence it is equivalent to the special 3-opt that does not reverse any edges. A local optimum with respect to the special 3-opt is a local optimum in problem space with respect to moving one city. This raises the following questions: Are there other heuristics for which a nice equivalence between local optimality in heuristic or problem space and another kind of local optimality occurs? Does the $\theta(n \log n)$ worst-case ratio of the space-filling curve improve after the special 3-opt is applied? That would be evidence for the usefulness of problem space search.

5. For a classical optimization problem such as bin packing, find a known heuristic whose worst-case performance improves with local optimality in heuristic space.

6. Prove theoretically and empirically that the ordering of variables affects the performance of algorithms that do crossover.

7. For the greedy nearest-neighbor TSP heuristic, conduct local search that changes one edge length until reaching a local optimum. What is the empirical computational performance? Find an upper bound on the ratio of local optimum tour length to optimal tour length. Estimate the number of local optima. Estimate the SA convergence parameter value. The same questions apply for the Euclidean TSP, moving one city.

8. Is there a deep relationship between smoothed analysis and search in problem space? Both ask how an algorithm will perform if the data are perturbed slightly. Does good smoothed performance predict good problem space search performance, if the problem space search is made in a Lin–Kernighan way (i.e., you don't perturb the perturbed)?

9. Many NIA articles jump from nonexistence of the second derivative to the necessity for their black-box derivative-free algorithms. But if the first derivative can be computed and is Lefschetz, a secant (quasi-Newton) method has superlinear convergence. Can computational performance on some of the functions be significantly improved?

10. Rigorously test how well or poorly successful NIAs scale up. Do they tend to scale up in a different way from other NIAs or other algorithms such as random restart local search?

8.3. Questions to Ask Ourselves

Section 8.2 poses questions for individual researchers. I count on the prevalent incentives—the intellectual challenge and the prospect of journal publications—to motivate people to work on them. My underlying goal is not to get an answer to any particular question; it is to get OR researchers to study NIAs with our traditionally high standards for mathematical analysis and computational testing. I see this as a necessary step toward merging OR with NIAs. In contrast, this section poses questions for the OR community as a whole. The issues I raise here cannot be tackled without leadership. Journal editors and officers in optimization societies could lead on issues of computational standards. Department heads and INFORMS officers could lead on structural issues such as incentives.

- How much of a threat or benefit is the NIA field to the OR community? What effect is it having on our market penetration, research funding, public visibility, consulting, software, and impact on the U.S. and global economies?
- Why have we given relatively little attention to continuous variable global nonlinear optimization? Has that changed over the years? Are our mathematical aesthetics the only cause?
- Can we find an objective standard by which we can judge global optimization, analogous to the size of the largest Euclidean TSP exactly solved?

- Should INFORMS pay someone to keep watch on the public and science media, and rebut false statements when appropriate?
- If we should engage with the NIA community, what incentives would work, and how can we create them?
- If we would benefit from more studies such as that of Sergeyev et al. [109], what incentives would work, and how can we create them?

9. Conclusions

Writing this tutorial has broadened my perspective. I hope that reading it has broadened yours. Here are a few concluding thoughts.

- The plethora of NIAs belies their many algorithmic similarities. To overload a metaphor, they have practiced horizontal gene transfer and experienced convergent evolution.
- In general, NIA research and applications require much less (human) time and technical expertise than does mathematical programming. Practitioners tend to prefer NIAs that are easy to learn and quick to implement, over computationally superior OR algorithms that have higher start-up costs. OR has failed to create tools as attractive and simple to use as NIAs.
- The academic OR community has, for the most part, ignored NIAs. I believe this is a mistake. Much of the science and engineering community is unaware of the large gap between what NIAs and mathematical programming can do on standard OR models. Even more troubling is the vast territory outside what standard OR models can accommodate, territory once vacant but now teeming with NIAs that threaten to engulf OR.

According to the published literature, there are many thousands of successful NIA applications, and most Fortune 500 companies are using NIAs daily. If so, we have work to do. Either our methods are not as good as we think they are or there are thousands of opportunities for us to improve an existing optimization application.

- Unconventional NIAs that search in spaces of problem data, algorithm parameters, and encodings have achieved some impressive successes. They merit more attention.
- I have suggested several steps we in OR could take now regarding the existence of the large metaheuristics community. That these steps have not already been taken implies that there has been no incentive to do so.

Acknowledgments

The author thanks Esma Gel and Doug Shier, who invited him to write this tutorial, for encouragement, help, and patience with delays. The author also thanks Anne Raymond, the late Stephen Gould, Tom Seeley, Bill Shields, Ivan Chase, Jeannette Yen, Marc Weissburg, and Janine Benyus for teaching him the little he knows about evolution. He also thanks Ted Pavlic and the reviewers, who added many insights and corrected many of the author's mistakes. Any errors that remain are the author's sole responsibility.

Endnotes

[1] One of the peculiarities of the NIA literature is a doggedness in seizing a metaphor in one's teeth and chewing on it to such a pulp that the reader can barely swallow it. When researchers observed that the best solutions in a population are sometimes lost because they get mutated, crossed, or passed over by the randomness in selection, they modified GA to identify and pass on the best or K-best solutions to the next population. The modification worked, but its computational success apparently was not satisfying without a metaphor. Thus "elitism" was born.

[2] One can understand why most of the math programming community dismissed GA research at the time, if they knew of it at all. The hardware used in the tests included a Sun 3/60, which had 20 MHz cycle speed and 24 MB core memory. Yet 10×10 was the maximum size the program could handle. By contrast, in the 1960s, a PDP 5 with 8 KB memory and 0.167 MHz cycle speed (six-microsecond cycle time) solved linear programming problems (LPs) larger than that for farmers (Calvin [16]). (These were feed formulation (diet problem) LPs.)

[3] Velocities are in units of distance per time step.

[4] They inaptly named it "inertia weight" because it is a numerical weight on the inertial term in the equation. In the physical movement analogy, it is fluidity (i.e., the opposite of viscosity).

[5]This may be compared with the 10,300 citations of Dantzig's book on linear programming and extensions (Dantzig [25]).

[6]Macready and Wolpert [74] find a flaw in Holland's bandit problem proof, and they challenge the correctness of the claimed result.

[7]According to Yang and Karamanoglu [132], heuristics, by definition, operate by trial and error. Some other NAI researchers' definitions are closer to the meaning in OR (Michalewicz and Fogel [82]).

[8]Holland demonstrated his intellectual integrity a few years later by challenging his own building block hypothesis and finding it wanting (Mitchell et al. [80, 81]).

[9]Nature abhors a vacuum.

References

[1] E. Aarts and J. K. Lenstra. *Local Search in Combinatorial Optimization*. Princeton University Press, Princeton, NJ, 2003.

[2] F. J. Ahlers, W. Di Carlo, C. Fleiner, L. Godwin, M. Keenan, R. Deb Nath, A. Neumaier, et al. Differential evolution (DE) for continuous function optimization by Kenneth Price and Rainer Storn. Accessed August 22, 2018, http://www1.icsi.berkeley.edu/~storn/code.html.

[3] S. Anily and A. Federgruen. Simulated annealing methods with general acceptance probabilities. *Journal of Applied Probability* 24(3):657–667, 1987.

[4] A. Banks, J. Vincent, and C. Anyakoha. A review of particle swarm optimization. Part I: Background and development. *Natural Computing* 6(4):467–484, 2007.

[5] A. Banks, J. Vincent, and C. Anyakoha. A review of particle swarm optimization. Part II: Hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing* 7(1):109–124, 2008.

[6] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende, and W. R. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics* 1(1):9–32, 1995.

[7] J. Bartholdi, T. Seeley, C. Tovey, and J. VandeVate. The pattern and effectiveness of forager allocation among food sources in honey bee colonies. *Journal of Theoretical Biology* 160(1):23–40, 1993.

[8] J. M. Benyus. *Biomimicry: Innovation Inspired by Nature*. Morrow, New York, 1997.

[9] D. Bertsimas and J. Tsitsiklis. Simulated annealing. *Statistical Science* 8(1):10–15, 1993.

[10] S. Binitha and S. S. Sathya. A survey of bio inspired optimization algorithms. *International Journal of Soft Computing and Engineering* 2(2):137–151, 2012.

[11] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys* 35(3):268–308, 2003.

[12] E. Bonabeau, M. Dorigo, and G. Theraulaz. Inspiration for optimization from social insect behaviour. *Nature* 406(6791):39–42, 2000.

[13] I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences* 237(July):82–117, 2013.

[14] H. J. Bremermann. Optimization through evolution and recombination. M. C. Yovits, G. T. Jacobi, and G. D. Goldstein, eds. *Self-Organizing Systems, 1962*. Spartan Books, Washington, DC, 93–106, 1962.

[15] K. M. Burjorjee. The fundamental problem with the building block hypothesis. Working paper, Brandeis University, Waltham, MA, https://arxiv.org/abs/0810.3356, 2008.

[16] D. Calvin. Oral communication, January 1985.

[17] R. G. Carter. Pipeline optimization: Dynamic programming after 30 years. *PSIG Annual Meeting*, Pipeline Simulation Interest Group, Houston, 1998.

[18] Center for Biologically Inspired Design. Center for Biologically Inspired Design (CBID) at Georgia Tech home page. Accessed August 22, 2018, http://www.cbid.gatech.edu.

[19] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications* 45(1):41–51, 1985.

[20] V. A. Cicirello and S. F. Smith. Ant colony control for autonomous decentralized shop floor routing. *Proceedings of the 5th International Symposium on Autonomous Decentralized Systems, 2001*. IEEE, Piscataway, NJ, 383–390, 2001.

[21] P. Civicioglu and E. Besdok. A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. *Artificial Intelligence Review* 39(4):315–346, 2013.

[22] W. Cook. Concorde TSP solver home page. Accessed August 22, 2018, http://www.math.uwaterloo.ca/tsp/concorde.html.

[23] B. J. Copeland and D. Proudfoot. Alan Turing's forgotten ideas in computer science. *Scientific American* 280(4):98–103, 1999.

[24] H. P. Crowder, R. S. Dembo, and J. M. Mulvey. Reporting computational experiments in mathematical programming. *Mathematical Programming* 15(1):316–329, 1978.

[25] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[26] K. A. DeJong. An analysis of the behavior of a class of genetic adaptive systems. Unpublished PhD thesis, University of Michigan, Ann Arbor, 1975.

[27] M. Dorigo. Optimization, learning and natural algorithms. (In Italian.) PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992.

[28] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1991.

[29] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. *Proceedings of the 2000 Congress on Evolutionary Computation*, Vol. 1. IEEE, Piscataway, NJ, 84–88, 2000.

[30] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science* (*MHS '95*). IEEE, Piscataway, NJ, 39–43, 1995.

[31] R. C. Eberhart, Y. Shi, and J. Kennedy. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, 2001.

[32] R. W. Eglese. Simulated annealing: A tool for operational research. *European Journal of Operational Research* 46(3):271–281, 1990.

[33] M. Eusuff, K. Lansey, and F. Pasha. Shuffled frog-leaping algorithm: A memetic meta-heuristic for discrete optimization. *Engineering Optimization* 38(2):129–154, 2006.

[34] I. Fister Jr, X.-S. Yang, I. Fister, J. Brest, and D.Fister. A brief review of nature-inspired algorithms for optimization. *Elektrotehniški Vestnik* 80(3):116–122, 2013.

[35] D. B. Fogel. *Evolutionary Computation: The Fossil Record*. Wiley-IEEE Press, Piscataway, NJ, 1998.

[36] L. J. Fogel, A. J. Owens, and M. J. Walsh. On the evolution of artificial intelligence (artificial intelligence generated by natural evolution process). *Proceedings of the 5th National Symposium on Human Factors in Electronics*, San Diego, 63–76, 1964.

[37] L. J. Fogel, A. J. Owens, and M. J. Walsh. Intelligent decision-making through a simulation of evolution. *Simulation* 5(4):267–279, 1965.

[38] T. Friedrich, T. Kötzing, and M. S. Krejca. EDAs cannot be balanced and stable. T. Friedrich, ed. *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM, New York, 1139–1146, 2016.

[39] A. H. Gandomi and A. H. Alavi. Krill herd: A new bio-inspired optimization algorithm. *Communications in Nonlinear Science and Numerical Simulation* 17(12):4831–4845, 2012.

[40] V. Gazi and K. M. Passino. Stability analysis of swarms. *IEEE Transactions on Automatic Control* 48(4):692–697, 2003.

[41] Z. W. Geem, J. H. Kim, and G. V. Loganathan. A new heuristic optimization algorithm: Harmony search. *Simulation* 76(2):60–68, 2001.

[42] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research* 13(5):533–549, 1986.

[43] F. Glover. Tabu search—Part I. *ORSA Journal on Computing* 1(3):190–206, 1989.

[44] F. Glover. Tabu search—Part II. *ORSA Journal on Computing* 2(1):4–32, 1990.

[45] F. Glover and M. Laguna. Tabu search. D.-Z. Du and P. M. Pardalos, eds. *Handbook of Combinatorial Optimization*. Springer, Boston, 2093–2229, 1998.

[46] D. E. Goldberg. Computer-aided pipeline operation using genetic algorithms and rule learning. Unpublished PhD thesis, University of Michigan, Ann Arbor, 1983.

[47] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[48] J. J. Grefenstette, ed. *Proceedings of an International Conference on Genetic Algorithms and Their Applications*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

[49] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics* 16(1):122–128, 1986.

[50] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research* 13(2): 311–329, 1988.

[51] R. Hassan, B. Cohanim, O. De Weck, and G. Venter. A comparison of particle swarm optimization and the genetic algorithm. *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, American Institute of Aeronautics and Astronautics, Reston, VA, 1897.

[52] T. C. Havens, C. J. Spain, N. G. Salmon, and J. M. Keller. Roach infestation optimization. *IEEE Swarm Intelligence Symposium* (*SIS 2008*). IEEE, Piscataway, NJ, 1–7, 2008.

[53] J. C. Holland. *Adaptation in Natural and Artificial Systems*, 1st ed. University of Michigan Press, Ann Arbor, 1975.

[54] J. C. Holland. Genetic algorithms. *Scientific American* 267(1):66–73, 1992.

[55] J N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics* 1(1):33–42, 1995.

[56] H. S. Hosseini. Problem solving by intelligent water drops. *IEEE Congress on Evolutionary Computation (CEC 2007)*. IEEE, Piscataway, NJ, 2007.

[57] H. S. Hosseini. The intelligent water drops algorithm: A nature-inspired swarm-based optimization algorithm. *International Journal of Bio-Inspired Computation* 1(1–2):71–79, 2009.

[58] M. Jamil and H.-J. Zepernick. Lévy flights and global optimization. X.-S. Yang, Z. Cui, R. Xiao, A. Gandomi, and M. Karamanoglu, eds. *Swarm Intelligence and Bio-Inspired Computation: Theory and Applications*. Elsevier, London, 49–72, 2013.

[59] M. Jamil, X.-S. Yang, and H.-J. Zepernick. Test functions for global optimization: A comprehensive survey. X.-S. Yang, Z. Cui, R. Xiao, A. Gandomi, and M. Karamanoglu, eds. *Swarm Intelligence and Bio-Inspired Computation: Theory and Applications*. Elsevier, London, 2013.

[60] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. *Operations Research* 37(6): 865–892, 1989.

[61] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research* 39(3):378–406, 1991.

[62] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga. A comprehensive survey: Artificial bee colony (ABC) algorithm and applications. *Artificial Intelligence Review* 42(1):21–57, 2014.

[63] J. Kennedy. Particle swarm optimization. C. Sammut and G. I. Webb, eds. *Encyclopedia of Machine Learning* Springer, New York, 760–766, 2011.

[64] J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. *1997 IEEE International Conference on Systems, Man, and Cybernetics: Computational Cybernetics and Simulation*, Vol. 5. IEEE, Piscataway, NJ, 4104–4108, 1997.

[65] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal* 49(2):291–307, 1970.

[66] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science* 220(4598):671–680, 1983.

[67] S. Koenig, B. Szymanski, and Y. Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence* 31(1–4):41–76, 2001.

[68] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[69] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal* 44(10):2245–2269, 1965.

[70] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research* 21(2):498–516, 1973.

[71] Y. Liu and K. M. Passino. Biomimicry of social foraging bacteria for distributed optimization: Models, principles, and emergent behaviors. *Journal of Optimization Theory and Applications* 115(3):603–628, 2002.

[72] Y. Liu, K. M. Passino, and M. M. Polycarpou. Stability analysis of *M*-dimensional asynchronous swarms with a fixed communication topology. *IEEE Transactions on Automatic Control* 48(1): 76–95, 2003.

[73] M. Lundy and A. Mees. Convergence of an annealing algorithm. *Mathematical Programming* 34(1):111–124, 1986.

[74] W. G. Macready and D. H. Wolpert. On 2-armed Gaussian bandits and optimization. Technical Report SFI-TR-96-03-009, Santa Fe Institute, Santa Fe, New Mexico, 1996.

[75] J. McCall. Genetic algorithms for modelling and optimisation. *Journal of Computational and Applied Mathematics* 184(1):205–222, 2005.

[76] A. R. Mehrabian and C. Lucas. A novel numerical optimization algorithm inspired from weed colonization. *Ecological Informatics* 1(4):355–366, 2006.

[77] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics* 21(6):1087–1092, 1953.

[78] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd ed. Springer-Verlag, Berlin, 1994.

[79] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer Science & Business Media, New York, 2013.

[80] M. Mitchell, S. Forrest, and J. H. Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, 245–254, 1992.

[81] M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hill climbing? J. D. Cowan, G. Tesauro, and J. Alspector, eds. *Advances in Neural Information Processing Systems* (*NIPS '93*). Morgan Kaufmann, San Francisco, 51–58, 1994.

[82] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli. Convergence and finite-time behavior of simulated annealing. *Advances in Applied Probability* 18(3):747–771, 1986.

[83] S. Nakrani and C. Tovey. On honey bees and dynamic server allocation in internet hosting centers. *Adaptive Behavior* 12(3–4):223–240, 2004.

[84] S. Nakrani and C. Tovey. From honey bees to internet servers: Biomimicry for distributed management of internet hosting centers. *Bioinspiration and Biomimetics* 2(4):S182–S197, 2007.

[85] P. S. Oliveto and C. Zarges. Analysis of diversity mechanisms for optimisation in dynamic environments with low frequencies of change. *Theoretical Computer Science A* 561(Part A):37–56, 2013.

[86] U.-M. O'Reilly and F. Oppacher. The troubling aspects of a building block hypothesis for genetic programming. L. D. Whitley and M. D. Vose, eds. *Foundations of Genetic Algorithms*, Vol. 3. Morgan Kaufmann, San Francisco, 73–88, 1995.

[87] B.L. Partridge. The structure and function of fish schools. *Scientific American* 246(6):114–123, 1982.

[88] K. M. Passino. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems* 22(3):52–67, 2002.

[89] T. Pavlic. Personal communication, email, June 18, 2018.

[90] M. Pawlyn. *Biomimicry in Architecture*. RIBA Publishing, Newcastle Upon Tyne, UK, 2016.

[91] R. Pfeifer, M. Lungarella, and F. Iida. Self-organization, embodiment, and biologically inspired robotics. *Science* 318(5853):1088–1093, 2007.

[92] D. T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi. The bees algorithm. Technical report, Manufacturing Engineering Centre, Cardiff University, Cardiff, UK, 2005.

[93] D. T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi. The bees algorithm—A novel tool for complex optimisation problems. D. T. Pham, E. E. Eldukhri, and A. J. Soroka, eds. *Intelligent Production Machines and Systems: 2nd I*PROMS Virtual International Conference*. Elsevier, Amsterdam, 454–459, 2006.

[94] R. Poli. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, 2008(January):Article 4, 2008.

[95] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization: An overview. *Swarm Intelligence* 1(1):33–57, 2007.

[96] R. D. Quinn, G. M. Nelson, R. J. Bachmann, D. A. Kingsley, J. T. Offi, T. J. Allen, and R. E. Ritzmann. Parallel complementary strategies for implementing biological principles into mobile robots. *International Journal of Robotics Research* 22(3–4):169–186, 2003.

[97] O. A. Raouf and I. M. Hezam. Sperm motility algorithm: A novel metaheuristic approach for global optimisation. *International Journal of Operational Research* 28(2):143–163, 2017.

[98] R. L. Rardin, C. A. Tovey, and M. G. Pilcher. Analysis of a random cut test instance generator for the TSP. P. M. Pardalos, ed. *Complexity in Numerical Optimization*. World Scientific, Singapore, 387–405, 1993.

[99] I. Rechenberg. Evolutionsstrategie—Optimierung technisher Systeme nach Prinzipien der biologischen Evolution (Evolution Strategy: Optimization of Technical systems by means of biological evolution). PhD thesis, Technical University of Berlin, 1971.

[100] C. Reeves. Genetic algorithms. F. W. Glover and G. A. Kochenberger, eds. *Handbook of Metaheuristics*, 1st ed. Springer, Berlin, 55–82, 2003.

[101] C. R. Reeves. Genetic algorithms. M. Gendreau and J.-Y. Potvin, eds. *Handbook of Metaheuristics*, 2nd ed. Springer, Berlin, 109–139, 2010.

[102] T. J. Richer and T. M. Blackwell. The Lévy particle swarm. *2006 IEEE Congress on Evolutionary Computation* (*CEC 2006*). IEEE, Piscataway, NJ, 808–815, 2006.

[103] M. Rosendo and A. Pozo. A hybrid particle swarm optimization algorithm for combinatorial optimization problems. *2010 IEEE Congress on Evolutionary Computation* (*CEC 2010*). IEEE, Piscataway, NJ, 1–8, 2010.

[104] M. P. Saka, E. Doğan, and I. Aydogdu. Analysis of swarm intelligence-based algorithms for constrained optimization. X.-S. Yang, Z. Cui, R. Xiao, A. Gandomi, and M. Karamanoglu, eds. *Swarm Intelligence and Bio-Inspired Computation: Theory and Applications*. Elsevier, London, 25–48, 2013.

[105] K. Sastry, D. E. Goldberg, and G. Kendall. Genetic algorithms. E. K. Burke and G. Kendall, eds. *Search Methodologies* Springer, New York, 93–117, 2014.

[106] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Chichester, UK, 1981.

[107] H.-P. Schwefel. *Evolution and Optimum Seeking*, Vol. 1515. John Wiley & Sons, New York, 1995.

[108] T. D. Seeley and C. A. Tovey. Why search time to find a food-storer bee accurately indicates the relative rates of nectar collecting and nectar processing in honey bee colonies. *Animal Behaviour* 47(2):311–316, 1994.

[109] Y. D. Sergeyev, D. E. Kvasov, and M. S. Mukhametzhanov. On the efficiency of nature-inspired metaheuristics in expensive global optimization with limited budget. *Scientific Reports* 8(1): Article 453, 2018.

[110] Y. Shi and R. Eberhart. A modified particle swarm optimizer. 1998 *IEEE International Conference on Evolutionary Computation Proceedings: IEEE World Congress on Computational Intelligence*. IEEE, Piscataway, NJ, 69–73, 1998.

[111] K. Sörensen, M. Sevaux, and F. Glover. A history of metaheuristics. R. Martí, P. Pardalos, and M. Resende, eds. *Handbook of Heuristics*. Springer International, Cham, Switzerland, 1–18, 2018.

[112] R. H. Storer, S. D. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science* 38(10):1495–1509, 1992.

[113] R. H. Storer, S. D. Wu, and R. Vaccari. Problem and heuristic space search strategies for job shop scheduling. *ORSA Journal on Computing* 7(4):453–467, 1995.

[114] R. Storn and K. Price. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* 11(4):341–359, 1997.

[115] D. Sudholt and C. Witt. Update strength in EDAs and ACO: How to avoid genetic drift. T. Friedrich, ed. *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM, New York, 61–68, 2016.

[116] S. Sun and H. Liu. Particle swarm algorithm: Convergence and applications. X.-S. Yang, Z. Cui, R. Xiao, A. Gandomi, and M. Karamanoglu, eds. *Swarm Intelligence and Bio-Inspired Computation*. Elsevier, London, 137–168, 2013.

[117] J. Svennebring and S. Koenig. Trail-laying robots for robust terrain coverage. *Proceedings of the IEEE International Conference on Robotics and Automation* (*ICRA'03*), Vol. 1. IEEE, Piscataway, NJ, 75–82, 2003.

[118] K. Tang, X. Yáo, P. N. Suganthan, C. MacNish, Y.-P. Chen, C.-M. Chen, and Z. Yang. *Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization*. Nature Inspired Computation and Applications Laboratory, University of Science and Technology of China, Heifei, China, 24, 2007.

[119] C. A. Tovey. Hill climbing with multiple local optima. *SIAM Journal on Algebraic Discrete Methods* 6(3):384–393, 1985.

[120] C. A. Tovey. Low order polynomial bounds on the expected performance of local improvement algorithms. *Mathematical Programming* 35(2):193–224, 1986.

[121] C. A. Tovey. Simulated simulated annealing. *Journal of Mathematics and Management Sciences* 8(3–4):389–407, 1988.

[122] I. C. Trelea. The particle swarm optimization algorithm: Convergence analysis and parameter selection. *Information Processing Letters* 85(6):317–325, 2003.

[123] J. N. Tsitsiklis. Markov chains with rare transitions and simulated annealing. *Mathematics of Operations Research* 14(1):70–90, 1989.

[124] A. Turing. Intelligent machinery. D. C. Ince, ed. *Collected Works of A.M. Turing: Mechanical Intelligence*. Elsevier Science, New York, 545–548, 1992.

[125] R. Vaccari. Novel space definitions and search methods for job shop scheduling. Unpublished master's thesis, Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, 1990.

[126] F. Van den Berghand and A. P. Engelbrecht. A convergence proof for the particle swarm optimiser. *Fundamenta Informaticae* 105(4):341–374, 2010.

[127] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1(1):67–82, 1997.

[128] P. J. Wong and R. E. Larson. Optimization of natural gas pipeline systems via dynamic programming. *IEEE Transactions of Automatic Control* 13(5):475–481, 1968.

[129] X.-S. Yang. Firefly algorithms for multimodal optimization. O. Watanabe and T. Zeugmann, eds. *Stochastic Algorithms: Foundations and Applications* (*SAGA 2009*), Lecture Notes in Computer Science, Vol. 579. Springer, Berlin, 169–178, 2009.

[130] X.-S. Yang. *Nature-Inspired Optimization Algorithms*. Elsevier, London, 2014.

[131] X.-S. Yang and S. Deb. Cuckoo search via Lévy flights. *World Congress on Nature and Biologically Inspired Computing* (*NaBIC 2009*). IEEE, Piscataway, NJ, 210–214, 2009.

[132] X.-S. Yang and M. Karamanoglu. Swarm intelligence and bio-inspired computation: An overview. X.-S. Yang, Z. Cui, R. Xiao, A. Gandomi, and M. Karamanoglu, eds. *Swarm Intelligence and Bio-Inspired Computation: Theory and Applications*. Elsevier, London, 3–23, 2013.