Available online at www.sciencedirect.com
SCIENCE DIRECT
Advanced Engineering Informatics 19 (2005) 43–53
ADVANCED ENGINEERING INFORMATICS
www.elsevier.com/locate/aei

Comparison among five evolutionary-based optimization algorithms

Emad Elbeltagi[a,*], Tarek Hegazy[b,1], Donald Grierson[b,2]

[a]Department of Structural Engineering, Faculty of Engineering, Mansoura University, Mansoura 35516, Egypt
[b]Department of Civil Engineering, University of Waterloo, Waterloo, Ont., Canada N2L 3G1

Received 15 October 2004; revised 9 January 2005; accepted 19 January 2005

Abstract

Evolutionary algorithms (EAs) are stochastic search methods that mimic the natural biological evolution and/or the social behavior of species. Such algorithms have been developed to arrive at near-optimum solutions to large-scale optimization problems, for which traditional mathematical techniques may fail. This paper compares the formulation and results of five recent evolutionary-based algorithms: genetic algorithms, memetic algorithms, particle swarm, ant-colony systems, and shuffled frog leaping. A brief description of each algorithm is presented along with a pseudocode to facilitate the implementation and use of such algorithms by researchers and practitioners. Benchmark comparisons among the algorithms are presented for both continuous and discrete optimization problems, in terms of processing time, convergence speed, and quality of the results. Based on this comparative analysis, the performance of EAs is discussed along with some guidelines for determining the best operators for each algorithm. The study presents sophisticated ideas in a simplified form that should be beneficial to both practitioners and researchers involved in solving optimization problems.

Keywords: Evolutionary algorithms; Genetic algorithms; Memetic algorithms; Particle swarm; Ant colony; Shuffled frog leaping; Optimization

1. Introduction

The difficulties associated with using mathematical optimization on large-scale engineering problems have contributed to the development of alternative solutions. Linear programming and dynamic programming techniques, for example, often fail (or reach local optimum) in solving NP-hard problems with large number of variables and non-linear objective functions [1]. To overcome these problems, researchers have proposed evolutionary-based algorithms for searching near-optimum solutions to problems.

Evolutionary algorithms (EAs) are stochastic search methods that mimic the metaphor of natural biological evolution and/or the social behavior of species. Examples include how ants find the shortest route to a source of food and how birds find their destination during migration. The behavior of such species is guided by learning, adaptation, and evolution [1]. To mimic the efficient behavior of these species, various researchers have developed computational systems that seek fast and robust solutions to complex optimization problems. The first evolutionary-based technique introduced in the literature was the genetic algorithms (GAs) [2]. GAs were developed based on the Darwinian principle of the 'survival of the fittest' and the natural process of evolution through reproduction. Based on its demonstrated ability to reach near-optimum solutions to large problems, the GAs technique has been used in many applications in science and engineering [3–5]. Despite their benefits, GAs may require long processing time for a near-optimum solution to evolve. Also, not all problems lend themselves well to a solution with GAs [6].

In an attempt to reduce processing time and improve the quality of solutions, particularly to avoid being trapped in local optima, other EAs have been introduced during the past 10 years. In addition to various GA improvements, recent developments in EAs include four other techniques inspired by different natural processes: memetic algorithms

* Corresponding author. Tel.: +20 50 224 4105; fax: +20 50 224 4690.
E-mail addresses: eelbelta@mans.edu.eg (E. Elbeltagi), tarek@uwaterloo.ca (T. Hegazy), grierson@uwaterloo.ca (D. Grierson).
[1] Tel.: +1 519 888 4567x2174; fax: +1 519 888 6197.
[2] Tel.: +1 519 888 4567x2412; fax: +1 519 888 6197.

1474-0346/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.aei.2005.01.004

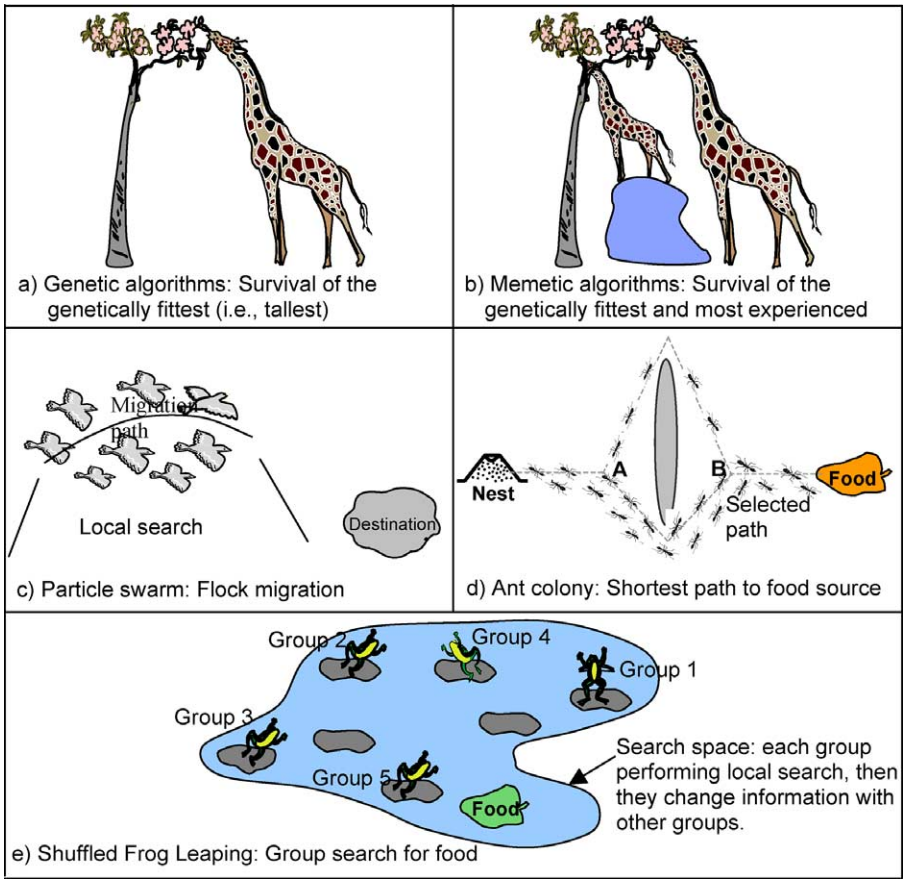

Fig. 1. Schematic diagram of natural evolutionary systems.

(MAs) [7], particle swarm optimization (PSO) [8], ant-colony systems [9], and shuffled frog leaping (SFL) [10]. A schematic diagram of the natural processes that the five algorithms mimic is shown in Fig. 1.

In this paper, the five EAs presented in Fig. 1 are reviewed and a pseudocode for each algorithm is presented to facilitate its implementation. Performance comparison among the five algorithms is then presented. Guidelines are then presented for determining the proper parameters to use with each algorithm.

2. Five evolutionary algorithms

In general, EAs share a common approach for their application to a given problem. The problem first requires some representation to suit each method. Then, the evolutionary search algorithm is applied iteratively to arrive at a near-optimum solution. A brief description of the five algorithms is presented in the following subsections.

2.1. Genetic algorithms

GAs are inspired by biological systems' improved fitness through evolution [2]. A solution to a given problem is represented in the form of a string, called 'chromosome', consisting of a set of elements, called 'genes', that hold a set of values for the optimization variables [11].

GAs work with a random population of solutions (chromosomes). The fitness of each chromosome is determined by evaluating it against an objective function. To simulate the natural survival of the fittest process, best chromosomes exchange information (through crossover or mutation) to produce offspring chromosomes. The offspring solutions are then evaluated and used to evolve the population if they provide better solutions than weak population members. Usually, the process is continued for a large number of generations to obtain a best-fit (near-optimum) solution. More details on the mechanism of GAs can be found in Goldberg [11] and Al-Tabtabai and Alex [3].

A pseudocode for the GAs algorithm is shown in Appendix A. Four main parameters affect the performance of GAs: population size, number of generations, crossover rate, and mutation rate. Larger population size (i.e. hundreds of chromosomes) and large number of generations (thousands) increase the likelihood of obtaining a global optimum solution, but substantially increase processing time.

Crossover among parent chromosomes is a common natural process [12] and traditionally is given a rate that

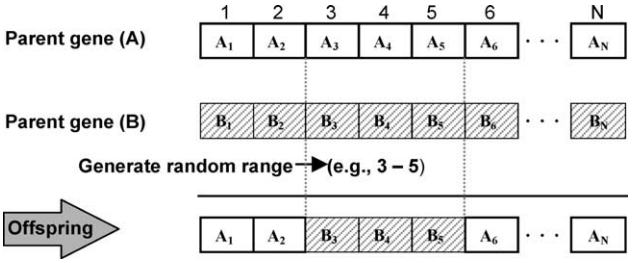

Fig. 2. Crossover operation to generate offspring.

ranges from 0.6 to 1.0. In crossover, the exchange of parents' information produces an offspring, as shown in Fig. 2. As opposed to crossover, mutation is a rare process that resembles a sudden change to an offspring. This can be done by randomly selecting one chromosome from the population and then arbitrarily changing some of its information. The benefit of mutation is that it randomly introduces new genetic material to the evolutionary process, perhaps thereby avoiding stagnation around local minima. A small mutation rate less than 0.1 is usually used [11].

The GA used in this study is steady state (an offspring replaces the worst chromosome only if is better than it) and real coded (the variables are represented in real numbers). The main parameters used in the GA procedure are population size, number of generations, crossover rate and mutation rate.

2.2. Memetic algorithms

MAs are inspired by Dawkins' notion of a *meme* [13]. MAs are similar to GAs but the elements that form a chromosome are called memes, not genes. The unique aspect of the MAs algorithm is that all chromosomes and offsprings are allowed to gain some experience, through a local search, before being involved in the evolutionary process [14]. As such, the term MAs is used to describe GAs that heavily use local search [15]. A pseudocode for a MA procedure is given in Appendix B.

Similar to the GAs, an initial population is created at random. Afterwards, a local search is performed on each population member to improve its experience and thus obtain a population of local optimum solutions. Then, crossover and mutation operators are applied, similar to GAs, to produce offsprings. These offsprings are then subjected to the local search so that local optimality is always maintained.

Merz and Freisleben [14] proposed one approach to perform local search through a pair-wise interchange heuristic (Fig. 3). In this method, the local-search neighborhood is defined as the set of all solutions that can be reached from the current solution by swapping two elements (memes) in the chromosome. For a chromosome of length n, the neighborhood size for the local search i:

$$N = 1/2 \times n \times (n-1) \quad (1)$$

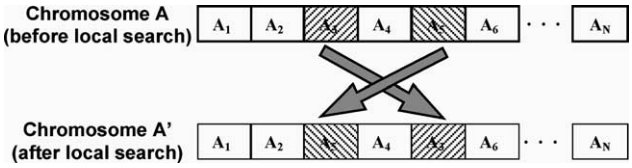

Fig. 3. Applying local search using pair-wise interchange.

The number of swaps and consequently the size of the neighborhood grow quadratically with the chromosome length (problem variables). In order to reduce processing time, Merz and Freisleben [14] suggested stopping the pair-wise interchange after performing the first swap that enhances the objective function of the current chromosome. The local-search algorithm, however, can be designed to suit the problem nature. For example, another local search can be conducted by adding or subtracting an incremental value from every gene and testing the chromosome's performance. The change is kept if the chromosome's performance improves; otherwise, the change is ignored. A pseudocode of this modified local search is given in Appendix C. As discussed, the parameters involved in MAs are the same four parameters used in GAs: population size, number of generations, crossover rate, and mutation rate in addition to a local-search mechanism.

2.3. Particle swarm optimization

PSO was developed by Kennedy and Eberhart [8]. The PSO is inspired by the social behavior of a flock of migrating birds trying to reach an unknown destination. In PSO, each solution is a 'bird' in the flock and is referred to as a 'particle'. A particle is analogous to a chromosome (population member) in GAs. As opposed to GAs, the evolutionary process in the PSO does not create new birds from parent ones. Rather, the birds in the population only evolve their social behavior and accordingly their movement towards a destination [16].

Physically, this mimics a flock of birds that communicate together as they fly. Each bird looks in a specific direction, and then when communicating together, they identify the bird that is in the best location. Accordingly, each bird speeds towards the best bird using a velocity that depends on its current position. Each bird, then, investigates the search space from its new local position, and the process repeats until the flock reaches a desired destination. It is important to note that the process involves both social interaction and intelligence so that birds learn from their own experience (local search) and also from the experience of others around them (global search).

The pseudocode for the PSO is shown in Appendix D. The process is initialized with a group of random particles (solutions), N. The ith particle is represented by its position as a point in a S-dimensional space, where S is the number of variables. Throughout the process, each particle i monitors

three values: its current position (X_i); the best position it reached in previous cycles (P_i); its flying velocity (V_i). These three values are represented as follows:

$$\left.\begin{array}{ll} \text{Current position} & X_i = (x_{i1}, x_{i2}, \ldots, x_{iS}) \\ \text{Best previous position} & P_i = (p_{i1}, p_{i2}, \ldots, p_{iS}) \\ \text{Flying velocity} & V_i = (v_{i1}, v_{i2}, \ldots, v_{iS}) \end{array}\right\} \quad (2)$$

In each time interval (cycle), the position (P_g) of the best particle (g) is calculated as the best fitness of all particles. Accordingly, each particle updates its velocity V_i to catch up with the best particle g, as follows [16]:

$$\text{New } V_i = \omega \times \text{current } V_i + c_1 \times \text{rand}(\,) \times (P_i - X_i) + c_2 \times \text{Rand}(\,) \times (P_g - X_i) \quad (3)$$

As such, using the new velocity V_i, the particle's updated position becomes:

$$\text{New position } X_i = \text{current position } X_i + \text{New } V_i; \qquad V_{\max} \geq V_i \geq -V_{\max} \quad (4)$$

where c_1 and c_2 are two positive constants named learning factors (usually $c_1 = c_2 = 2$); rand( ) and Rand( ) are two random functions in the range [0, 1], $V_{\max}$ is an upper limit on the maximum change of particle velocity [8], and ω is an inertia weight employed as an improvement proposed by Shi and Eberhart [16] to control the impact of the previous history of velocities on the current velocity. The operator ω plays the role of balancing the global search and the local search; and was proposed to decrease linearly with time from a value of 1.4–0.5 [16]. As such, global search starts with a large weight and then decreases with time to favor local search over global search [17].

It is noted that the second term in Eq. (3) represents *cognition*, or the private thinking of the particle when comparing its current position to its own best. The third term in Eq. (3), on the other hand, represents the *social* collaboration among the particles, which compares a particle's current position to that of the best particle [18]. Also, to control the change of particles' velocities, upper and lower bounds for velocity change is limited to a user-specified value of $V_{\max}$. Once the new position of a particle is calculated using Eq. (4), the particle, then, flies towards it [16]. As such, the main parameters used in the PSO technique are: the population size (number of birds); number of generation cycles; the maximum change of a particle velocity $V_{\max}$; and ω.

2.4. Ant-colony optimization

Similar to PSO, ant-colony optimization (ACO) algorithms evolve not in their genetics but in their social behavior. ACO was developed by Dorigo et al. [9] based on the fact that ants are able to find the shortest route between their nest and a source of food. This is done using pheromone trails, which ants deposit whenever they travel, as a form of indirect communication.

As shown in Fig. 1d, when ants leave their nest to search for a food source, they randomly rotate around an obstacle, and initially the pheromone deposits will be the same for the right and left directions. When the ants in the shorter direction find a food source, they carry the food and start returning back, following their pheromone trails, and still depositing more pheromone. As indicated in Fig. 1d, an ant will most likely choose the shortest path when returning back to the nest with food as this path will have the most deposited pheromone. For the same reason, new ants that later starts out from the nest to find food will also choose the shortest path. Over time, this positive feedback (autocatalytic) process prompts all ants to choose the shorter path [19].

Implementing the ACO for a certain problem requires a representation of S variables for each ant, with each variable i has a set of n_i options with their values l_{ij}, and their associated pheromone concentrations $\{\tau_{ij}\}$; where $i = 1, 2, \ldots, S$, and $j = 1, 2, \ldots, n_i$. As such, an ant is consisted of S values that describe the path chosen by the ant as shown in Fig. 4 [20]. A pseudocode for the ACO is shown in Appendix E. Other researchers use a variation of this general algorithm, incorporating a local search to improve the solution [21].

In the ACO, the process starts by generating m random ants (solutions). An ant k ($k = 1, 2, \ldots, m$) represents a solution string, with a selected value for each variable. Each ant is then evaluated according to an objective function. Accordingly, pheromone concentration associated with each possible route (variable value) is changed in a way to reinforce good solutions, as follows [9]:

$$\tau_{ij}(t) = \rho\tau_{ij}(t-1) + \Delta\tau_{ij}; \quad t = 1, 2, \ldots, T \quad (5)$$

where T is the number of iterations (generation cycles); $\tau_{ij}(t)$ is the revised concentration of pheromone associated with option l_{ij} at iteration t, $\tau_{ij}(t-1)$ is the concentration of pheromone at the previous iteration ($t-1$); $\Delta\tau_{ij}$ = change in pheromone concentration; and ρ = pheromone evaporation rate (0–1). The reason for allowing pheromone evaporation is to avoid too strong influence of the old pheromone to avoid premature solution stagnation [22]. In Eq. (5), the change in

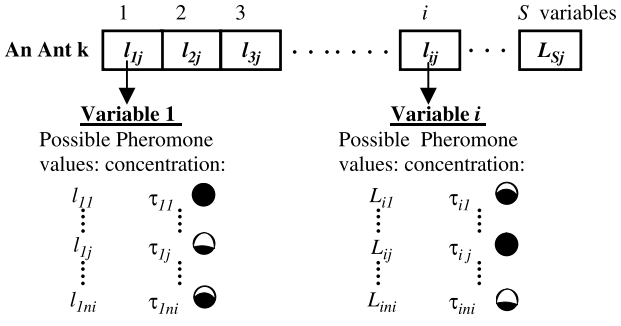

Fig. 4. Ant representation.

pheromone concentration $\Delta\tau_{ij}$ is calculated as [9]:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \begin{cases} R/\text{fitness}_k & \text{if option } l_{ij} \text{ is chosen by ant } k \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

where R is a constant called the pheromone reward factor; and fitness_k is the value of the objective function (solution performance) calculated for ant k. It is noted that the amount of pheromone gets higher as the solution improves. Therefore, for minimization problems, Eq. (6) shows the pheromone change as proportional to the inverse of the fitness. In maximization problems, on the other hand, the fitness value itself can be directly used.

Once the pheromone is updated after an iteration, the next iteration starts by changing the ants' paths (i.e. associated variable values) in a manner that respects pheromone concentration and also some heuristic preference. As such, an ant k at iteration t will change the value for each variable according to the following probability [9]:

$$P_{ij}(k,t) = \frac{[\tau_{ij}(t)]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{l_{ij}} [\tau_{ij}(t)]^{\alpha} \times [\eta_{ij}]^{\beta}} \quad (7)$$

where $P_{ij}(k,t)$ = probability that option l_{ij} is chosen by ant k for variable i at iteration t; $\tau_{ij}(t)$ = pheromone concentration associated with option l_{ij} at iteration t; η_{ij} = heuristic factor for preferring among available options and is an indicator of how good it is for ant k to select option l_{ij} (this heuristic factor is generated by some problem characteristics and its value is fixed for each option l_{ij}); and α and β are exponent parameters that control the relative importance of pheromone concentration versus the heuristic factor [20]. Both α and β can take values greater than zero and can be determined by trial and error. Based on the previous discussion, the main parameters involved in ACO are: number of ants m; number of iterations t; exponents α and β; pheromone evaporation rate ρ; and pheromone reward factor R.

2.5. Shuffled frog leaping algorithm

The SFL algorithm, in essence, combines the benefits of the genetic-based MAs and the social behavior-based PSO algorithms. In the SFL, the population consists of a set of frogs (solutions) that is partitioned into subsets referred to as memeplexes. The different memeplexes are considered as different cultures of frogs, each performing a local search. Within each memeplex, the individual frogs hold ideas, that can be influenced by the ideas of other frogs, and evolve through a process of memetic evolution. After a defined number of memetic evolution steps, ideas are passed among memeplexes in a shuffling process [23]. The local search and the shuffling processes continue until defined convergence criteria are satisfied [10].

As described in the pseudocode of Appendix F, an initial population of P frogs is created randomly. For S-dimensional problems (S variables), a frog i is represented as $X_i = (x_{i1}, x_{i2}, \ldots, x_{iS})$. Afterwards, the frogs are sorted in a descending order according to their fitness. Then, the entire population is divided into m memeplexes, each containing n frogs (i.e. $P = m \times n$). In this process, the first frog goes to the first memeplex, the second frog goes to the second memeplex, frog m goes to the mth memeplex, and frog $m+1$ goes back to the first memeplex, etc.

Within each memeplex, the frogs with the best and the worst fitnesses are identified as X_b and X_w, respectively. Also, the frog with the global best fitness is identified as X_g. Then, a process similar to PSO is applied to improve only the frog with the worst fitness (not all frogs) in each cycle. Accordingly, the position of the frog with the worst fitness is adjusted as follows:

$$\text{Change in frog position } (D_i) = \text{rand}(\,) \times (X_b - X_w) \quad (8)$$

$$\text{New position } X_w = \text{current position } X_w + D_i; \quad D_{\max} \geq D_i \geq -D_{\max} \quad (9)$$

where rand( ) is a random number between 0 and 1; and $D_{\max}$ is the maximum allowed change in a frog's position. If this process produces a better solution, it replaces the worst frog. Otherwise, the calculations in Eqs. (8) and (9) are repeated but with respect to the global best frog (i.e. X_g replaces X_b). If no improvement becomes possible in this case, then a new solution is randomly generated to replace that frog. The calculations then continue for a specific number of iterations [10]. Accordingly, the main parameters of SFL are: number of frogs P; number of memeplexes; number of generation for each memeplex before shuffling; number of shuffling iterations; and maximum step size.

3. Comparison among evolutionary algorithms' results

All the EAs described earlier have been coded using the Visual Basic programming language and all experiments took place on a 1.8 GHz AMD Laptop machine. The performance of the five EAs is compared using two benchmark problems for continuous optimization and a third problem for discrete optimization. A description of these test problems is given in the following.

3.1. Continuous optimization

Two well-known continuous optimization problems are used to test four of the EAs: *F8* (Griewank's) function and the *F10* function. Details of these functions are as follows.

3.1.1. F8 (Griewank's function)

The objective function to be optimized is a scalable, non-linear, and non-separable function that may take any

number of variables (x_is), i.e.

$$f(x_{i|i=1,N}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} (\cos(x_i/\sqrt{i})) \quad (10)$$

The summation term of the *F8* function (Eq. (10)) includes a parabolic shape while the cosine function in the product term creates waves over the parabolic surface. These waves create local optima over the solution space [24]. The *F8* function can be scaled to any number of variables *N*. The values of each variable are constrained to a range (−512 to 511). The global optimum (minimum) solution for this function is known to be zero when all *N* variables equal zero.

3.1.2. F10 function

This function is non-linear, non-separable, and involves two variables *x* and *y*, i.e.

$$f10(x, y) = (x^2 + y^2)^{0.25}[\sin^2(50(x^2 + y^2)^{0.1}) + 1] \quad (11)$$

To scale this function (Eq. (11)) to any number of variables, an extended *EF10* function is created using the following relation, [24],

$$EF(x_{i|i=N}) = \sum_{j=1}^{N} \sum_{i=1}^{N} F(x_i, x_j) \quad (12)$$

Accordingly, the extended *F10* function is:

$$EF10(x_{i|i=1,N}) = \sum_{i=1}^{N-1} (x_i^2 + x_{i+1}^2)^{0.25}[\sin^2(50(x_i^2 + x_{i+1}^2)^{0.1}) + 1] \quad (13)$$

Similar to the *F8* function, the global optimum solution for this function is known to be zero when all *N* variables equal zero, for the variable values ranging from −100 to 100.

3.2. Discrete optimization

In this section, a time-cost trade-off (TCT) construction management problem is used to compare among the five EAs with respect to their ability to solve discrete optimization problems. The problem relates to an 18-activity construction project that was described in Ref. [25]. The activities, their predecessors, and durations are presented in Table 1 along with five optional methods of construction that vary from cheap and slow (option 5) to fast and expensive (option 1). The 18 activities were input to a project management software (Microsoft Project) with activity durations being set to those of option 5 (least costs and longest durations among the five options). The total direct cost of the project in this case is $99,740 (sum of all activities' costs for option 5) with the project duration being 169 days (respecting the precedence relations in Table 1). The indirect cost of $500/day was then added to obtain a total project cost of $184,240.

With the initial schedule exceeding a desired deadline of 110-days, it is required to search for the optimum set of construction options that meet the deadline at minimum total cost. In this problem, the decision variables are the different methods of construction possible for each activity (i.e. five discrete options, 1–5, with associated durations and costs). The objective function is to minimize the total project cost (direct and indirect) and is formulated

Table 1
Test problem for discrete optimization

Activity no.	Depends on	Option 1		Option 2		Option 3		Option 4		Option 5	
		Duration (days)	Cost ($)	Duration (days)	Cost ($)	Duration (days)	Cost ($)	Duration (days)	Cost ($)	Duration (days)	Cost ($)
1	–	14	2400	15	2150	16	1900	21	1500	24	1200
2	–	15	3000	18	2400	20	1800	23	1500	25	1000
3	–	15	4500	22	4000	33	3200	–	–	–	–
4	–	12	45,000	16	35,000	20	30,000	–	–	–	–
5	1	22	20,000	24	17,500	28	15,000	30	10,000	–	–
6	1	14	40,000	18	32,000	24	18,000	–	–	–	–
7	5	9	30,000	15	24,000	18	22,000	–	–	–	–
8	6	14	220	15	215	16	200	21	208	24	120
9	6	15	300	18	240	20	180	23	150	25	100
10	2, 6	15	450	22	400	33	320	–	–	–	–
11	7, 8	12	450	16	350	20	300	–	–	–	–
12	5, 9, 10	22	2000	24	1750	28	1500	30	1000	–	–
13	3	14	4000	18	3200	24	1800	–	–	–	–
14	4, 10	9	3000	15	2400	18	2200	–	–	–	–
15	12	12	4500	16	3500	–	–	–	–	–	–
16	13, 14	20	3000	22	2000	24	1750	28	1500	30	1000
17	11, 14, 15	14	4000	18	3200	24	1800	–	–	–	–
18	16, 17	9	3000	15	2400	18	2200	–	–	–	–

as follows:

$$\text{Min}\left(T \times I + \sum_{i=1}^{n} C_{ij}\right) \tag{14}$$

where n = number of activities; C_{ij} = direct cost of activity i using its method of construction j; T = total project duration; and I = daily indirect cost. To facilitate the optimization using the different EAs, macro programs of the 5 EAs were written using the VBA language that comes with the Microsoft Project software. The data in Table 1 were stored in one of the tables associated with the software. When any one of the EA routines is activated, the evolutionary process selects one of the five construction options to set the activities' durations and costs. Accordingly, the project's total cost (objective function) and duration changes. The evolutionary process then continues to attempt to optimize the objective function.

3.3. Parameter settings for evolutionary algorithms

As discussed earlier, each algorithm has its own parameters that affect its performance in terms of solution quality and processing time. To obtain the most suitable parameter values that suit the test problems, a large number of experiments were conducted. For each algorithm, an initial setting of the parameters was established using values previously reported in the literature [4,8,10,16,20]. Then, the parameter values were changed one by one and the results were monitored in terms of the solution quality and speed. The final parameter values adopted for each of the five EAs are given in the following.

3.3.1. Genetic algorithms

The crossover probability (C_P) and the mutation probability (M_P) were set to 0.8 and 0.08, respectively. The population size was set at 200 and 500 offsprings. The evolutionary process was kept running until no improvements were made in the objective function for 10 consecutive generation cycles (i.e. 500×10 offsprings) or the objective function reached its known target value, whichever comes first.

3.3.2. Memetic algorithms

MAs are similar to GAs but apply local search on chromosomes and offsprings. The standard pair-wise interchange search does not suit the continuous functions *F8* and *F10*, and the local-search procedure in Appendix C is used instead. For the discrete problem, on the other hand, the pair-wise interchange was used. The same values of $C_P = 0.8$ and $M_P = 0.08$ that were used for the GAs are applied to the MAs. After experimenting with various values, a population size of 100 chromosomes was used for the MAs.

3.3.3. Particle swarm optimization

Upon experimentation, the suitable numbers of particles and generations were found to be 40 and 10,000, respectively. Also, the maximum velocity was set as 20 for the continuous problems and 2 for the discrete problem. The inertia weight factor ω was also set as a time-variant linear function decreasing with the increase of number of generations where, at any generation i,

$$\omega = 0.4 + 0.8 \times (\text{number of generations} - i)/(\text{number of generations} - 1) \tag{15}$$

such that $\omega = 1.2$ and 0.4 at the first and last generation, respectively.

3.3.4. Ant-colony optimization

As the ACO algorithm is suited to discrete problems alone, no experiments were done using it for the *F8* and *F10* test functions. However, the TCT discrete problem was used for experimentation with the ACO. After extensive experimentation, 30 ants and 100 iterations were found suitable. Also, the other parameters were set as follows: $\alpha = 0.5$; $\beta = 2.5$; ρ (pheromone evaporation rate) = 0.4; and R (reward factor depends on problem nature) = 10.

3.3.5. Shuffled frog leaping

Different settings were experimented with to determine suitable values for parameters to solve the test problems using the SFL algorithm. A population of 200 frogs, 20 memeplexes, and 10 iterations per memeplex were found suitable to obtain good solutions.

3.4. Results and discussions

The results found from solving the three test problems using the five EAs, which represents a fairly wide class of problems, are summarized in Tables 2 and 3, and Fig. 5 (the Y axis of Fig. 5 is a log scale to show long computer run times). It is noted that the processing time for solving the *EF10* function was similar to that of the *F8* function and follows the same trend as shown in Fig. 5.

Twenty trial runs were performed for each problem. The performance of the different algorithms was compared using three criteria: (1) the percentage of success, as represented by the number of trials required for the objective function to reach its known target value; (2) the average value of the solution obtained in all trials; (3) the processing time to reach the optimum target value. The processing time, and not the number of generation cycles, was used to measure the speed of each EA, because the number of generations in each evolutionary cycle is different from one algorithm to another. In all experiments, the solution stopped when one of two following criteria was satisfied: (1) the *F8* and *EF10* objective functions reached a target value of 0.05 or less (i.e. to within an acceptable tolerance of the known optimum value of zero), or 110 days for the TCT problem;

Table 2
Results of the continuous optimization problems

Comparison criteria	Algorithm	Number of variables						
		F8				*EF10*		
		10	20	50	100	10	20	50
% Success	GAs (Evolver)	50	30	10	0	20	0	0
	MAs	90	100	100	100	100	70	0
	PSO	30	80	100	100	100	80	60
	ACO	–	–	–	–	–	–	–
	SFL	50	70	90	100	80	20	0
Mean solution	GAs (Evolver)	0.06	0.097	0.161	0.432	0.455	1.128	5.951
	MAs	0.014	0.013	0.011	0.009	0.014	0.068	0.552
	PSO	0.093	0.081	0.011	0.011	0.009	0.075	2.895
	ACO	–	–	–	–	–	–	–
	SFL	0.08	0.063	0.049	0.019	0.058	2.252	6.469

Table 3
Results of the discrete optimization problem

Algorithm	Minimum project duration (days)	Average project duration (days)	Minimum cost ($)	Average cost ($)	% Success rate	Processing time (s)
GAs	113	120	162,270	164,772	0	16
MAs	110	114	161,270	162,495	20	21
PSO	110	112	161,270	161,940	60	15
ACO	110	122	161,270	166,675	20	10
SFL	112	123	162,020	166,045	0	15

or (2) the objective function value did not improve in ten consecutive generations. To experiment with different problem sizes, the *F8* test function in Eq. (10) was solved using 10, 20, 50, and 100 variables, while the *EF10* test function in Eq. (13) was solved using 10, 20, and 50 variables (it becomes too complex for larger numbers of variables).

Surprisingly, the GA performed more poorly than all the other four algorithms. In fact, it was found to perform more poorly than even that reported in Whitley et al. [24] and Raphael and Smith [26] when using the CHC and Genitor GAs, while it performed better than the ESGAT GA version. A commercial GA package, Evolver [27], was used to verify the results. Evolver is an add-in program to Microsoft Excel, where the objective function, variables (adjustable cells), and the constraints are readily specified by highlighting the corresponding spreadsheet cells. Evolver performed almost the same way as the VB code with slight improvement. The results of using Evolver are reported in Table 2. The difference in GA results than those reported in Refs. [24,26] may in part be because the GA utilized in this paper uses real rather than binary coding.

As shown in Table 2 for the *F8* function, the GA was able to reach the target for 50% of the trials with 10 variables, and the number of successes decreased as the number of variables increased. Despite its inability to reach the optimum value of zero with the larger number of 100 variables, the GA was able to achieve a solution close to the optimum (0.432 for the *F8* function with 100 variables). Also, it is noticed from Fig. 5 that as the number of variables increased, the processing time to reach the target also increased (from 5 min:12 s with 10 variables to 40 min:27 s with 50 variables). As shown in Table 2 for the *EF10* test function, the GA was only able to achieve 20% success using 10 variables, and that the solution quality decreased as the number of variables increased (e.g. the objective function = 5.951 using 50 variables). Using the GA to solve the TCT problem, the minimum solution obtained was 113 days with a minimum total cost of $162,270 and the success rate for reaching the optimum solution was zero, as shown in Table 3.

Upon applying the MA, the results improved significantly compared to those obtained using the GA, in terms of both the success rate (Table 2) and the processing time (Fig. 5). Solving the *F8* function using 100 variables, for example,

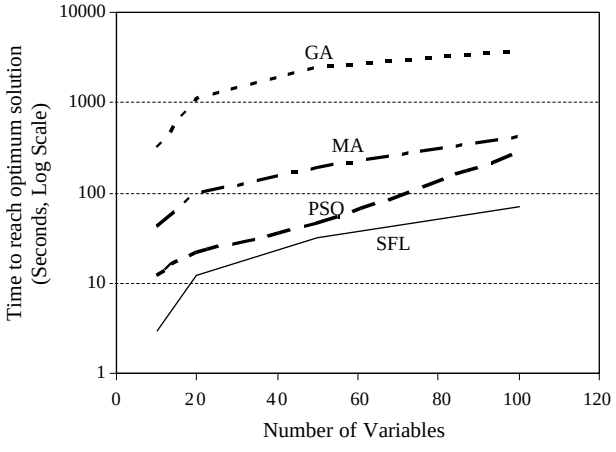

Fig. 5. Processing time to reach the optimum for *F8* function.

the success rate was 100% with a processing time of 7 min:08 s. Even for the trials with less success rate, as shown in Table 2, the solutions were very close to the optimum. That is to say, the local search of the MA improved upon the performance of the GA. When applying the MA to the TCT problem, it was able to reach the optimum project duration of 110 days and a total cost of $161,270, with a 20% success rate and an average cost that improved upon that of the GA (Table 3). It is to be noted that the local-search module presented in Appendix C was applied for the *F8* and *EF8* functions, while the pair-wise interchange local-search module was applied to the TCT problem.

The PSO algorithm outperformed the GA and the MA in solving the *EF10* function in terms of the success rate (Table 2), the processing time (Fig. 5), while it was less successful than the MA in solving the *F8* function. Also, the PSO algorithm outperformed all other algorithms when used to solve the TCT problem, with a success rate of 60% and average total cost of $161,940, as shown in Table 3.

The ACO algorithm was applied only to the TCT discrete optimization problem. While it was able to achieve the same success rate as the GA (20%), the average total cost of the 20 runs was greater than that of all other algorithms (Table 3). This is due to the scattered nature of the obtained results (minimum duration of 110 days, and maximum duration of 139 days) caused by premature convergence that happened in some runs. To avoid premature convergence, the pair-wise interchange local-search module was applied and the results obtained were greatly improved with a success rate of 100%, but the average processing time increased from 10 to 48 s.

When solving the *F8* and *EF10* test functions using the SFL algorithm, it was found that the success rate (Table 2) was better than the GA and similar to that for PSO. However, it performed less well when used to solve the *EF10* function. As shown in Fig. 5, the SFL processing times were the least among all algorithms. Interestingly, it is noticed from Table 2 that as the number of variables increased for the *F8* function, the success rates for SFL, MA and PSO all increased. This is because the *F8* function becomes smoother as its dimensions increase [24]. As opposed to this trend, the success rate decreased for the GA as the number of variables increased. The same trend for the GA was also reported in Refs. [24,26] when used to solve the *F8* function. Also, using the SFL algorithm to solve the TCT problem, the minimum duration obtained was 112 days with minimum total cost of $162,020 (Table 3). While the success rate for the SFL was zero, its performance was better than the GA.

It is interesting to observe that the behavior of each optimization algorithm in all test problems (continuous and discrete) was consistent. In particular, the PSO algorithm generally outperformed all other algorithms in solving all the test problems in terms of solution quality (except for the *F8* function with 10 and 50 variables). Accordingly, it can be concluded that the PSO is a promising optimization tool, in part due to the effect of the inertia weight factor ω. In fact, to take advantage of the fast speed of the SFL algorithm, the authors suggest using a weight factor in Eq. (3) for SFL that is similar to that used for PSO (some preliminary experiments conducted by the authors in this regard have shown good results).

4. Conclusions

In this paper, five evolutionary-based search methods were presented. These include: GA, MA, PSO, ACO, and SFL. A brief description of each method is presented along with a pseudocode to facilitate their implementation. Visual Basic programs were written to implement each algorithm. Two benchmark continuous optimization test problems were solved using all but the ACO algorithm, and the comparative results were presented. Also presented were the comparative results found when a discrete optimization test problem was solved using all five algorithms. The PSO method was generally found to perform better than other algorithms in terms of success rate and solution quality, while being second best in terms of processing time.

Appendix A. Pseudocode for a GA procedure

```
Begin;
  Generate random population of P solutions (chromosomes);
  For each individual i∈P: calculate fitness (i);
    For i=1 to number of generations;
      Randomly select an operation (crossover or mutation);
      If crossover;
        Select two parents at random i_a and i_b;
        Generate on offspring i_c=crossover (i_a and i_b);
      Else If mutation;
        Select one chromosome i at random;
        Generate an offspring i_c=mutate (i);
      End if;
      Calculate the fitness of the offspring i_c;
      If i_c is better than the worst chromosome then
      replace the worst chromosome by i_c;
    Next i;
  Check if termination=true;
End;
```

Appendix B. Pseudocode for a MA procedure

```
Begin;
  Generate random population of P solutions (chromosomes);
  For each individual i∈P: calculate fitness (i);
  For each individual i∈P: do local-search (i);
```

```
    For i = 1 to number of generations;
      Randomly select an operation (crossover or mutation);
      If crossover;
        Select two parents at random i_a and i_b;
        Generate on offspring i_c = crossover (i_a and i_b);
        i_c = local-search (i_c);
      Else If mutation;
        Select one chromosome i at random;
        Generate an offspring i_c = mutate (i);
        i_c = local-search (i_c);
      End if;
      Calculate the fitness of the offspring;
      If i_c is better than the worst chromosome then replace the worst chromosome by i_c;
    Next i;
  Check if termination = true;
End;
```

Appendix C. Pseudocode for the memetic local search

```
Begin;
  Select an incremental value d = a*Rand( ), where a is a constant that suits the variable values;
  For a given chromosome i ∈ P: calculate fitness (i);
  For j = 1 to number of variables in chromosome i;
    Value (j) = value (j) + d;
    If chromosome fitness not improved then value (j) = value (j) − d;
    If chromosome fitness not improved then retain the original value (j);
  Next j;
End;
```

Appendix D. Pseudocode for a PSO procedure

```
Begin;
  Generate random population of N solutions (particles);
  For each individual i ∈ N: calculate fitness (i);
  Initialize the value of the weight factor, ω;
    For each particle;
      Set pBest as the best position of particle i;
      If fitness (i) is better than pBest;
      pBest(i) = fitness (i);
    End;
    Set gBest as the best fitness of all particles;
    For each particle;
      Calculate particle velocity according to Eq. (3);
      Update particle position according to Eq. (4);
    End;
    Update the value of the weight factor, ω;
  Check if termination = true;
End;
```

Appendix E. Pseudocode for an ACO procedure

```
Begin;
  Initialize the pheromone trails and parameters;
    Generate population of m solutions (ants);
    For each individual ant k ∈ m: calculate fitness (k);
    For each ant determine its best position;
    Determine the best global ant;
    Update the pheromone trail;
  Check if termination = true;
End;
```

Appendix F. Pseudocode for a SFL procedure

```
Begin;
  Generate random population of P solutions (frogs);
  For each individual i ∈ P: calculate fitness (i);
  Sort the population P in descending order of their fitness;
    Divide P into m memeplexes;
    For each memeplex;
      Determine the best and worst frogs;
      Improve the worst frog position using Eqs. (4) or (5);
      Repeat for a specific number of iterations;
    End;
    Combine the evolved memeplexes;
    Sort the population P in descending order of their fitness;
  Check if termination = true;
End;
```

References

[1] Lovbjerg M. Improving particle swarm optimization by hybridization of stochastic search heuristics and self-organized criticality. Masters Thesis, Aarhus Universitet, Denmark; 2002.
[2] Holland J. Adaptation in natural and artificial systems. Ann Arbor, MI: University of Michigan Press; 1975.
[3] Al-Tabtabai H, Alex PA. Using genetic algorithms to solve optimization problems in construction. Eng Constr Archit Manage 1999;6(2):121–32.
[4] Hegazy T. Optimization of construction time-cost trade-off analysis using genetic algorithms. Can J Civil Eng 1999;26: 685–97.
[5] Grierson DE, Khajehpour S. Method for conceptual design applied to office buildings. J Comput Civil Eng 2002;16(2):83–103.
[6] Joglekar A, Tungare M. Genetic algorithms and their use in the design of evolvable hardware. http://www.manastungare.com/articles/genetic/genetic-algorithms.pdf; 2003, accessed on May 20, 2004, 15 p.

[7] Moscato P. On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Technical Report Caltech Concurrent Computation Program, Report 826, California Institute of Technology, Pasadena, CA; 1989.
[8] Kennedy J, Eberhart R. Particle swarm optimization. Proceedings of the IEEE international conference on neural networks (Perth, Australia), 1942–1948. Piscataway, NJ: IEEE Service Center; 1995.
[9] Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern 1996;26(1):29–41.
[10] Eusuff MM, Lansey KE. Optimization of water distribution network design using the shuffled frog leaping algorithm. J Water Resour Plan Manage 2003;129(3):210–25.
[11] Goldberg DE. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley Publishing Co; 1989.
[12] Caudill M. Evolutionary neural networks. AI Expert 1991;March: 28–33.
[13] Dawkins R. The selfish gene. Oxford: Oxford University Press; 1976.
[14] Merz P, Freisleben B. A genetic local search approach to the quadratic assignment problem. In: Bäck CT, editor. Proceedings of the 7th international conference on genetic algorithms. San Diego, CA: Morgan Kaufmann; 1997. p. 465–72.
[15] Moscato P, Norman MG. A memetic approach for the traveling salesman problem—implementation of a computational ecology for combinatorial optimization on message-passing systems. In: Valero M, Onate E, Jane M, Larriba JL, Suarez B, editors. International conference on parallel computing and transputer application. Amsterdam, Holland: IOS Press; 1992. p. 177–86.
[16] Shi Y, Eberhart R. A modified particle swarm optimizer. Proceedings of the IEEE international conference on evolutionary computation. Piscataway, NJ: IEEE Press; 1998. p. 69–73.
[17] Eberhart R, Shi Y. Comparison between genetic algorithms and particle swarm optimization. Proceedings of the 7th annual conference on evolutionary programming. Berlin: Springer; 1998. p. 611–8.
[18] Kennedy J. The particle swarm: social adaptation of knowledge. Proceedings of the IEEE international conference on evolutionary computation (Indianapolis, Indiana). Piscataway, NJ: IEEE Service Center; 1997. p. 303–8.
[19] Dorigo M, Gambardella LM. Ant colonies for the traveling salesman problem. Biosystems, Elsevier Sci 1997;43(2):73–81.
[20] Maier HR, Simpson AR, Zecchin AC, Foong WK, Phang KY, Seah HY, et al. Ant colony optimization for design of water distribution systems. J Water Resour Plan Manage 2003;129(3):200–9.
[21] Rajendran C, Ziegler H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of Jobs. Eur J Oper Res 2004;155(2):426–38.
[22] Merkle D, Middendorf M, Schmeck H. Ant colony optimization for resource-constrained project scheduling. Proceedings of the genetic and evolutionary computation conference (GECCO-2000); 2000. p. 893–900.
[23] Liong S-Y, Atiquzzaman Md. Optimal design of water distribution network using shuffled complex evolution. J Inst Eng, Singapore 2004;44(1):93–107.
[24] Whitley D, Beveridge R, Graves C, Mathias K. Test driving three genetic algorithms: new test functions and geometric matching. J Heurist 1995;1:77–104.
[25] Feng C, Liu L, Burns S. Using genetic algorithms to solve construction time-cost trade-off problems. J Comput Civil Eng 1997;11(3):184–9.
[26] Raphael B, Smith IFC. A direct stochastic algorithm for global search. J Appl Math Comput 2003;146(2/3):729–58.
[27] Evolver, Evolver Version 4.0.2, Palisade Corporation; 1998.