Evolutionary Optimization (EvOpt): A Brief Review and Analysis

Ruhul Sarker*, Joarder Kamruzzaman** and Charles Newton*
*School of ITEE, University of New South Wales, ADFA Campus, Canberra 2600, Australia
**GSCIT, Monash University, Churchill, Victoria, Australia
Email: ruhul@cs.adfa.edu.au

Abstract:
Evolutionary Computation (EC) has attracted increasing attention in recent years, as powerful computational techniques, for solving many complex real-world problems. The Operations Research (OR)/Optimization community is divided on the acceptability of these techniques. One group accepts these techniques as potential heuristics for solving complex problems and the other rejects them on the basis of their weak mathematical foundations. In this paper, we discuss the reasons for using EC in optimization. A brief review of Evolutionary Algorithms (EAs) and their applications is provided. We also investigate the use of EAs for solving a two-stage transportation problem by designing a new algorithm. The computational results are analysed and compared with conventional optimization techniques.

1. Introduction

Over the last two decades, Evolutionary Computation (EC), which is based on the principle of evolution (survival of the fittest), has shown tremendous success in solving complex real-world problems (Fischer and Leung, 2001 and Barnett et al., 2000). Engineers and scientists with quite different backgrounds have worked together to tackle some of the most difficult problems using a promising set of stochastic search algorithms - Evolutionary Algorithms (EAs). Evolutionary computation is one of the current hot topics in the area of computer science and optimization. Within the last decade, solving optimization problems using EAs has become an important application area of the evolutionary computation field. Due to its parallelism and some intelligent properties such as self-organization, adaptation and self-learning, EC has been applied successfully to many problems where the classical approaches are either un-available or generally lead to unsatisfactory results. In recent years, the interest in EC has been growing dramatically (Yao, 1999 and Patel et al., 2001).

The great success of EC was first recognized in the 1980s, when extremely complex optimization problems from various disciplines were solved, thus facilitating the undeniable breakthrough of EC as a problem-solving methodology (Goldberg, 1989). This breakthrough has been reflected through the growing number of publications in this field and a corresponding increase in specialized conferences and journals. Despite these conferences and journals, a large amount of application-specific work is widely scattered throughout the publications of many diverse disciplines, including Operations Research (OR) / Optimization, and presented at their related conferences, thus reflecting the general applicability and success of EC methods.

Solving optimization problems using evolutionary computation is known as Evolutionary Optimization (EvOpt). The huge popularity of EvOpt, in many other disciplines, encourages us to draw it to the OR / optimization research community’s attention. As we have experienced, the OR / Optimization researchers and practitioners are divided on the use of EC methods for solving optimization problems. OR / Optimization researchers and practitioners use, very often, heuristics to find near optimal solutions for many large and complex problems. EC methods are basically heuristics. EC methods do not have any strong mathematical basis, as the conventional optimization techniques have, and it is difficult to guarantee the convergence of the algorithms (for all problems) in a finite number of steps. In this paper, we introduce EC, in general, and analyse the similarities and differences to conventional search techniques. We discuss the reasons for using EC in optimization and the situations where EC could be useful. A brief review of the different EAs and their applications is provided. We also investigate the use of EAs for solving optimization problems. A new genetic algorithm has been developed for solving a two-stage transportation problem. The developed GA is
discussed briefly, and the computational results are analysed and compared with the conventional optimization techniques.

The organization of the paper is as follows: following this introduction, this paper briefly introduces evolutionary computation as an optimization problem solving tool. In Section 3, different EAs are discussed briefly, and the issue of using EC in optimization is explained in Section 4. Section 5 discusses the situations where EC could be useful in optimization. In Section 6, the disadvantages of using EC are presented. We discuss the use of EAs for solving optimization problems and a new GA for solving a two-stage transportation problem in Section 7. The computational experiences are also presented in Section 7. Finally, conclusions are drawn in section 8.

2. What is EC?

Evolutionary computation is the study of computational systems which uses ideas from natural evolution and adaptation. Many evolutionary computation techniques get their ideas and inspirations from molecular evolution, population genetics, immunology, etc. Some of the terminology used in evolutionary computation has been borrowed from these fields to reflect their connections, such as genetic algorithms, crossover, mutation, genotypes, phenotypes, species, etc. Although the research into evolutionary computation could help us understand some biological phenomena better, its primary aim is not to build biologically plausible models. The primary aim is to study and develop robust and efficient computational systems for solving complex real-world problems.

Evolutionary algorithms (EAs) are known as stochastic search procedures. Before we describe the general framework of EAs, let us present the general framework of a conventional, known as generate-and-test, search algorithm (Yao, 2002).

1. Generate an initial solution and denote it as the current solution;
2. Generate the next solution from the current one by perturbation;
3. Test whether the newly generated solution is acceptable;
   (a) If yes, accept it as the current solution;
   (b) Otherwise keep the current solution unchanged.
4. Go to Step (2) if the current solution is not satisfactory; stop otherwise.

Different algorithms use different strategies for the generation of the initial solution, the perturbation and acceptance of the current solution.

EAs have two prominent features which distinguish themselves from other search algorithms. First, they are all population-based (consider a set of possible solution points in contrast to single solution point in conventional optimization). Second, there is communications and information exchange among individuals in a population. A general framework of evolutionary algorithms can be summarised as follows.

1. Set $i = 0$;
2. Generate the initial population $P(i)$ at random;
3. REPEAT
   (a) Evaluate the fitness of each individual in $P(i)$;
   (b) Select parents from $P(i)$ based on their fitness;
   (c) Apply search operators to the parents and produce generation $P(i+1)$;
4. UNTIL the population converges or the maximum time is reached

EAs can be regarded as a population-based version of generate-and-test search. They use search operators like crossover and mutation to generate new solutions, and use selection to test which solutions are better. The search operators and the selection procedures will be discussed in a later section. It can be noted here that there is no strict rule to use crossover and mutation to generate new solutions in evolutionary computation (Yao, 2002). In principle, one may use any search procedure to generate new solutions that will increase the probability of finding a global optimum. This is also true for selection. This gives an opportunity for researchers, in Operations Research and Optimization, to contribute to the field of evolutionary computation.
Evolutionary computation can be divided into four major branches which are briefly discussed in the following section.

3. Different EAs: A Brief Introduction

Evolutionary computation encompasses four major branches, i.e., evolution strategies, evolutionary programming, genetic algorithms and genetic programming, due to historical reasons. At the philosophical level, they differ mainly in the level at which they simulate evolution. At the algorithmic level, they differ mainly in their representations of potential solutions and the operators used to modify the solutions. From a computational point of view, representation and search are two key issues. In this paper, we will briefly look at their differences at the algorithmic level.

Before we discuss the different evolutionary algorithms, it is appropriate to indicate that there exist different representations, search operators and selection procedures. Different EAs use different representations for the individuals. A good representation will make a problem easier to solve and a poor representation will do the opposite. There are three categories of representations: Lists, Trees and Graphs. For mathematical optimization problems, we are mainly interested in the lists representation. Lists contain a number of different methods such as binary strings, real-valued vectors, integer vectors, symbolic (character or digit) strings, etc. The details of these representations can be found in Michalewicz (1996). Representation is always best designed in conjunction with search operators. Search operators are the key players in the reproduction of offspring. The search operators used in the literature are listed below.

- **Crossover/Recombination**: k-point crossover, uniform crossover, intermediate crossover, global discrete crossover, order-based recombinations, matrix-based recombinations, etc.

- **Mutation**: bit-flipping, Gaussian mutation, Cauchy mutation, random generation of subtree, etc.

A selection scheme determines the probability of an individual being selected to produce offspring by recombination and/or mutation. There are three major types of selection schemes, roulette wheel selection (also known as the fitness proportional selection), rank-based selection (linear and nonlinear) and tournament selection. Sometimes elitist selection is used. Further details on search operators and selection can be found in Yao (2002).

Now let us discuss the different evolutionary algorithms. Evolution strategies (ES) were first proposed by Rechenberg and Schwefel in 1965 as a numerical optimization technique. The original evolution strategy did not use populations. A population was introduced into evolution strategies later (Schwefel, 1981 & 1995). In ES, real-valued vectors are used to represent individuals rather than binary strings. ES usually use a deterministic selection scheme, Gaussian mutation, and discrete or intermediate recombination. The term crossover is seldom used in the context of ES because ES do not simulate evolution at the genetic level. Further details on ES can be found in Schwefel (1981 & 1995).

Evolutionary programming (EP) was first proposed by Fogel and others in the mid 1960's as one way to achieve artificial intelligence (Fogel et al., 1966). Several examples of evolving finite state machines were demonstrated. Since late 1980's, evolutionary programming was also applied to various combinatorial and numerical optimization problems. EP is very similar to ES in terms of algorithm. It uses real numbers as individuals, Gaussian mutation and self-adaptation. The most noticeable differences between EP and ES are recombination and selection. EP does not use any recombination or crossover, but uses a probabilistic competition (i.e., a kind of tournament selection) as the selection mechanism. Further details on EP can be found in Fogel (1995). We like to mention here that there is no reason why EP cannot have recombination and why ES cannot have a probabilistic selection scheme from the algorithmic point of view.

Evolutionary algorithms (GAs) was first proposed by Holland (1975) and his student (Jong, 1975) in 1975 although some of the ideas appeared as early as 1957 in the context of simulating genetic systems (Fraser, 1957). Genetic algorithms were first proposed as adaptive search algorithms, although they have mostly been used as a global optimization algorithm for either combinatorial or numerical problems. They are probably the most well-known branch of evolutionary computation. GAs are quite different from ES and EP in terms of individual representation and search
operators. GAs emphasise genetic encoding of potential solutions into chromosomes and apply genetic operators to these chromosomes. This is equivalent to transforming the original problem from one space to another space. The genetic representation is crucial to the success of GAs. Although a simple GA uses binary representation, one point crossover, bit-flipping mutation and roulette-wheel selection, there appeared many variations in the literature. In GAs, the recombination (crossover) plays a major role and the mutation is only used as a background operator. Further details on GAs can be found in Michalewicz (1996).

A special sub-branch of genetic algorithms is Genetic Programming (GP). GP can be regarded as an application of genetic algorithms to evolve tree-structured chromosomes (Koza, 1992). Historically, those trees represent LISP programs. In GP, both crossover and mutation are used. Since GP is not usually used in mathematical optimization, we are not interested in GP.

In recent years, a general term of evolutionary algorithms has been used by more and more researchers to include all three major algorithms, i.e., evolution strategies, evolutionary programming and genetic algorithms, since they use almost the same computational framework. This is the view taken by this paper.

4. Why use EC in Optimization?

EC has been recognised as a great tool for solving optimization problems among Computer Science (CS) and Information Technology (IT) researchers over the last two decades. However, OR / Optimization people find it difficult to accept EC as a suitable tool for optimization mainly due to lack of its strong mathematical basis for convergence and optimality. In this section, the suitability of using EC for optimization will be explored indicating the advantages and disadvantages from the optimization point of view.

EC has some advantages over the conventional mathematical programming techniques. As our brief discussions on advantages below, one can easily imagine why EC should be used in solving optimization problems.

Properties of Functions
Consideration of convexity/ concavity and continuity of functions are not necessary in EC, however, these are a real concern in most mathematical programming techniques. In EC, the initial population (which is a set of solutions) is generated randomly and then the subsequent generations (a new set of solutions) are usually produced numerically (using simple rules) from their previous populations. It does not matter whether the function is differentiable or not. Although both the domains share the same concept of generate-and-test algorithm, the EC search procedure is not directly comparable (no clear similarity) to the conventional search, even without using derivatives, such as the cyclic coordinate method, the method of Hooke and Jeeves, and Rosenbrock’s method (for these methods see Bazaraa and Shetty, 1979). However, there is no restriction on incorporating any conventional search procedure into EC (say hybrid EC) to make the algorithm efficient.

Single Best Solution
The mathematical programming techniques provide a single best solution that is not the interest of many decision makers. For example, in the bidding problem, the decision maker is interested to see other solution alternatives such as the second best, third best and so on. This is due to the fact that the solution of the optimization problem is usually used as one of many bid evaluation criteria (Seydel and Olson, 1990). The second best, third best and so on solutions can be obtained easily by using EC. The solutions can also be manipulated to fit the decision maker's desire and qualitative goals.

Infeasibility
The mathematical programming techniques cannot, apparently, help in decision making when the problem is infeasible. However, the incorporation of penalty functions allows the identification of the infeasible constraints and hence helps to remove infeasibility by revising the constraints. EC is helpful in making the problem feasible by suggesting minimum changes in problem structure. It is common practice in EC to use penalty function based methods for all types of constrained optimization problems including linear programming (Sarker et al., 2001a), although other methods such as
repairing infeasible solutions (Whitely et al., 1996), and rejecting infeasible solutions (Michalewicz and Schmidt, 2001) are sometimes used.

Domain Knowledge
It is not difficult to implement EAs, because they do not require any rich domain knowledge. However, domain knowledge can be incorporated into EC techniques (Yao, 2002). For example, the concept of conventional search techniques can replace the search component in EAs. In addition, the final EAs solutions can be refined using local search techniques. The generation of the initial population and the selection procedure can be redesigned using the conventional optimization concepts.

Robustness of EC Algorithms
Theoretically, one common structure of evolutionary algorithms can be implemented for many single objective constrained mathematical programming models. Any single penalty function-based evolutionary algorithm can be used to solve many linear, integer and different classes of nonlinear programming models (Sarker et al, 2001a & 2001b). However, the algorithm may require different algorithmic parameters to solve different models. On the other hand, we need to learn a number of different and complex techniques to solve these models using conventional optimization theory. For example, the simplex method and interior point method for linear programs (Hillier and Lieberman, 2001 and Sierksma, 1996), and the branch and bound method and cutting plane method for integer programs (Martin, 1999). There are many different methods available for different classes of nonlinear programs (Eiselt et al, 1987).

Constrained Handling and Penalty Methods
The implementation of the penalty function method in EC is somewhat efficient. In conventional optimization, we solve the modified problem (unconstrained version with penalty) repeatedly changing the penalty parameters (Bazaraa and Shetty, 1979). In EC, we solve the modified problem only once and the penalty parameters are changed, using different rules, as the generation progresses (Sarker et al, 2001).

Exploration and Exploitation
EAs are a class of general purpose (domain independent) search methods which strike a remarkable balance between exploration and exploitation of the search space (see p-15, Michalewicz, 1996). This property helps to improve the solution by skipping from the local optima, and is particularly useful when solving multi-modal problems.

Computational Time
The most favourable point of EAs is that they provide quick approximate solutions. In most cases, EAs make a significant improvement within the first few generations (Varela, et al., 2003). For many hard problems, EC based techniques provide near-optimal solutions within reasonable computational time (Sakawa and Kato, 2003).

Multiobjective Optimization
EAs are more suitable for multiobjective optimization because of their capability of simultaneous optimization of conflicting objective functions and generation of a number of alternative solutions in a single run (Sarker et al., 2002, Zitzler et al., 2000 and Zitzler and Thiele, 1999). Some multi-objective problems can be solved, using conventional optimization techniques, developing a composite objective where the linear combination of weights (assigned to the individual objective functions) provides the alternative solutions for the Pareto-frontier (Miettinen, 1999). These problems usually belong to a class of problems which generate continuous and monotonically increasing and decreasing Pareto frontiers. However, if the Pareto-frontier is non-continuous (e.g., a set of discretely spaced continuous sub-regions) or non-uniform (higher density of solutions in one region than others), or the functions are complex, periodic or multimodal, then EAs are the better option (Deb, 1999).

Starting Solution
Unlike conventional search techniques, EC methods do not require any specialized methodology to generate the initial solutions (for example; North-west Corner Rule for the Transportation Problem). The initial population is usually generated randomly in EC.
Harder Problems
In conventional optimization domains, integer programming is more complex than linear programming, and nonlinear programming is even harder. In EC domains, hard problems are not that hard, compared to easy problems. When solving optimization problems using EC, the integer requirements, combinatorial nature, nonlinearity and other properties do not add much complexity (in general terms) compared to the linear programming problem. However, solving LP by using EC is not appropriate because of the power of existing conventional optimization algorithms. In case of multi-modal problems, there are less chances of being trapped in the local optima (Liang, 2000) when using EC.

Optimization under Changing and Dynamic Environments
For many real-world optimization problems, the environment fluctuates, leading to dramatic changes in the fitness (objective function values) of individual solutions. Examples of such problems include: on-line data mining – where the contents of the data-warehouse may change over time; job scheduling – where the problem is continually changing as jobs are completed and new jobs are added; investment portfolio evaluation – where the assessment of risks vary over time; and Robot’s path determination – where the new path must be determined for the next move (Wide and Schellwat, 1997). On-line Taxi-cab assignment problems (Kosoresow and Johnson, 2002) and Pizza delivery assignment problems are also considered as problems under changing environments where the optimal solutions are changing with time. Optimization under changing environments (dynamic or non-stationary or on-line) can be handled nicely by EC techniques (De Jong, 1999; Grefenstette, 1999 and Morrison and De Jong 1999). These problems can be handled to some extent considering a number of static instances.

5. When Should We Use Evolutionary Algorithms?
The power of existing linear programming algorithms would strongly discourage any use of EC based methodologies for solving linear programming problems. However, EC based methods could be beneficial for many real-world complex problems. We have identified the following cases where EC can contribute significantly better to the field of optimization and operations research.

Knowledge of Optimization: As discussed earlier, the researchers / practitioners with poor or no mathematical knowledge on optimization problems can still solve the problems using EC based methodology.

Ranked Solutions: When the decision makers requires 2nd best solution, 3rd best solution and so on, the EC based methodology could be a better option since they can produced all these solutions in a single run.

Multi-modal Problems: When the problem is multi-modal with many peaks, the EC based methodology has a lower probability of being trapped in local optima.

Quick Approximate Solutions: When we need quick approximate solutions to large-scale difficult problems, EC based methodology can do a better job in many situations.

Multi-objective Optimization: Multi-objective problems, where simultaneous optimization of multiple conflicting objectives is required, is a potential application area where EC based methodology can do better.

Optimization under Changing Environments: Optimization under a changing environment is another potential application / research area for EC methodology.

Hybrid Algorithms: When designing hybrid algorithms for solving complex problems, many features from conventional techniques or other modern techniques can be incorporated into EC to design more efficient algorithms.

Computationally Cheaper: EC based methods would be a preferred option, if they are computationally cheaper in solving any class of problems.
Computational time: If the computational time is not a concern, the EC based methods can be used to provide near optimal solutions.

Highly complex problem: Problems involving many complex features like multi-objectivity, multimodality, changing environment, etc would be suitable for EC techniques.

6. Disadvantages of Using EC

There are several drawbacks to using EC based methodology for solving optimization problems. They are briefly discussed below.

Heuristics: EC based techniques are recognized as heuristic search algorithms. There is no guarantee of optimality when solving optimization problems using such techniques (Sarker et al, 2001a & 2001b).

Parameters of EC: The parameters used in EC based techniques are problem oriented. It is not an easy task to determine the appropriate parameters for a given problem (Eiben et al., 1999).

Convergence of EAs: Convergence pattern and time complexity, while using EC based techniques for solving optimization problems, are case oriented too. Although some bounds are derived on the convergence rate of GAs, for specific cases, these can not be generalized for all the problems and the GAs applied to these problems (He and Kang, 1999; Takahashi, 1998 & Hanne, 1999).

Mathematical Insight: EAs do not help to examine the mathematical insight of the optimization problems to be solved which could provide additional information to the decision makers.

Sensitivity Analysis: Sensitivity analysis may be performed, for all type of models, on the desired range though it would not be as efficient as LP sensitivity.

7. EAs for Solving Optimization Problems

In the Operations Research (Optimization) domain, we judge the performance of algorithms, when solving optimization problems, based on (at least) two basic criteria: (i) the quality of solutions and (ii) the computational time (complexity). As per the literature, although there appeared a huge number of applications of EAs to the optimization problems, almost all researchers attempted to show (i) how the problem can be solved using EAs and (ii) their algorithms’ superiority (compared to similar classes of algorithms) based on the quality of solutions only (Zitzler et al., 2000; Thangiah and Nygard, 1992). In some cases, they compared the average number of fitness evaluated. It is hard to find papers where the authors compared their algorithms with the existing conventional optimization techniques based on both the criteria discussed above. This could be either due to simple ignorance or because of the fact that they are not computationally superior compared to other existing techniques. Whatever the reason – this does not help to motivate OR researchers to use EAs for solving optimization problems.

Now the question arises: is there any situation where EAs provide good solutions and are also computationally superior? As we pointed out earlier, solving LP using EAs would not be the right choice. Koksalan and Keha (2003) solved a number of single-machine bicriteria scheduling problems using GAs. They showed that GA outperformed Simulated Annealing (SA) for all the test problems in terms of quality of solutions. However, SA requires much shorter computational time than GA. Varlea et al. (2003) used GAs for solving job shop scheduling problems with bottlenecks. For a given problem, they have compared the solutions produced by three different generations (Table 7, Varlea et al., 2003). The percentage deviations of the fitness from the known best solution are 0.8330%, 0.0969% and 0.01259% for the number of generations 50, 200 and 1000 respectively. The first 50 generations provide a very good approximate solution (within 1%) which is really a favourable point for using GAs for quick approximate solutions. It is well-known in EAs that the most improvement is made within the first few generations and it is very slow after 50 generations.

Chu and Beasley (1997) developed a genetic algorithm for solving the generalised assignment problem, a well-known NP-complete combinatorial optimization problem. They mainly investigated
the quality of solutions produced rather than the computational expense. Although GA is superior in terms of solution quality, the computational times for some heuristics may be significantly shorter than those given for GA. Beasley and Chu (1996) claimed similar results with GA for a set covering problem which has no known polynomial time algorithms.

The integer programming problems are known as hard problems in the Operations Research / Optimization literature. Sakawa and Kato (2003) proposed a GA for solving such problems and compare the performance with the branch-and-bound (B&B) technique. In one set of experimental studies, they considered five instances of a multidimensional 0-1 knapsack problem with three different degrees of strictness of the constraints. The sizes of the problem instances are: (i) 30 variables and 10 constraints, (ii) 50 variables and 20 constraints, (iii) 100 variables and 30 constraints, (iv) 150 variables and 40 constraints and (v) 200 variables and 50 constraints. We compare the objective function values and the computational times, of these problem instances, for a degree of strictness of the constraints equal to 0.75. The details of parameters used for GA, LP package, computer platform and other information can be found in Sakawa and Kato (2003). As shown in the table, GAs are providing optimal solutions for smaller problems (problem instances 1 & 2) and the quality of solution is deteriorating (slowly) as the size of the problem instance increases. However, the computational time, for GA compared to B&B, is more and more appreciable as long as the size of the problem instance increases. This is a very favourable point for using EAs in solving hard optimization problems. However, further experimentation is required to generalize the findings.

Table 1: Comparing GA and Branch & Bound technique for integer programming

<table>
<thead>
<tr>
<th>Problem Instance</th>
<th>Objective Values</th>
<th>Computational Time (sec)</th>
<th>Objective Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (VxC)</td>
<td>Optimal</td>
<td>Best GA</td>
<td>B&amp;B</td>
</tr>
<tr>
<td>1 30x10</td>
<td>-12051</td>
<td>-12051</td>
<td>9.00x10^2</td>
</tr>
<tr>
<td>2 50x20</td>
<td>-21931</td>
<td>-21931</td>
<td>8.91x10^4</td>
</tr>
<tr>
<td>3 100x30</td>
<td>-46198</td>
<td>-46097</td>
<td>1.13x10^5</td>
</tr>
<tr>
<td>4 150x40</td>
<td>-66543</td>
<td>-65989</td>
<td>4.44x10^7</td>
</tr>
<tr>
<td>5 200x50</td>
<td>-86757</td>
<td>-85169</td>
<td>1.10x10^9</td>
</tr>
</tbody>
</table>

As we can see in Table 1, the computational times required by GAs for the problem instances 1, 2, 3, 4 and 5 are 69.33, 16.50, 0.422, 0.021 and 0.028 times, respectively, of the computational time required by the B&B technique.

7.1 A Two Stage Transportation Problem

Consider a two stage transportation problem where $n$ sources and $m$ intermediate / exchange points and $k$ destinations exist. The sources have limited supply, the exchange points have limited capacity to handle the supplied goods and the destinations have known demand. All the supplies from the sources are transported to the destinations via the exchange points. The problem is to minimize the overall transportation cost while satisfying supply, capacity, demand and flow constraints. This problem can be formulated as a simple linear program (LP) as follows.

**Notations:**

- $x_{ij}$: The quantity transported from source $i$ to exchange point $j$
- $x_{jk}$: The quantity transported from exchange point $j$ to destination $k$
- $C_{ij}$: Unit transportation cost from source $i$ to exchange point $j$
- $C_{jk}$: Unit transportation cost from exchange point $j$ to destination $k$
- $S_i$: Total supply from source $i$
- $CP_j$: Capacity of exchange point $j$
- $D_k$: Demand of destination $k$
Mathematical Model:

Minimize \[ Z = \sum_{i} \sum_{j} C_{ij}x_{ij} + \sum_{j} \sum_{k} C_{jk}x_{jk} \]

\[ \sum_{j} x_{ij} \leq S_i \]
\[ \sum_{i} x_{ij} \leq CP_j \]
\[ \sum_{j} x_{jk} \geq D_k \]
\[ \sum_{j} x_{ij} - \sum_{k} x_{jk} \geq 0 \]
\[ x_{ij}, x_{jk} \geq 0 \]

The objective of the problem is to minimize the overall transportation costs. The first, second and third constraints ensure the supply limitation, exchange point capacity and demand requirements respectively. The fourth or the flow constraint indicates that the total supply in the second stage of the network must be less than or equal to the supply in the first stage.

Although the quantity transported (decision variables of the model) requires integer values, the structure of the LP can ensure integer values for variables, provided supply, capacity and demand data are integer, without having any integer variables in the model. In this case study, we intend to solve this model using GA. From the previous study, it is found that the quality of solutions, while solving by GAs, deteriorates with the tightness of the constraints (Sarker et al., 2001a & 2001b). We have chosen this problem mainly for two reasons: (i) this problem generates many tighter constraints which is not an easy test problem for GA application and (ii) the mathematical model of this problem can be converted from a simple linear program to a difficult integer programming test problem with a very minor modification.

7.2 GA Application

First we consider a simple real-coded GA, where the variable values were generated as real/integer numbers not as binary-string. Usually the binary coding is recognized as the most suitable encoding for any problem solution because it maximizes the number of schemata being searched implicitly (Holland, 1975 and Goldberg, 1989), but there have been many examples in the evolutionary computation literature where alternative representations have resulted in algorithms with greater efficiency and optimization effectiveness when applied to identical problems (see e.g. articles by Back and Schwefel, 1993 and Fogel and Stayton, 1994, among others). Davis (1991) and Michalewicz (1996) comment that for many applications real-values or other representations may be chosen to advantage over binary coding. There does not appear to be any general benefit in maximizing implicit parallelism in evolutionary algorithms, and, therefore, forcing problems to fit into a binary representation may not be recommended. As experimented by Sarker et al (2001a & b), real coded GAs perform better than Binary coded GAs for integer programming problems.

The following characteristics were considered in running the Simple GA (SGA):

Algorithm 1: Simple GA

- Population size 50
- Initial population was generated randomly within the given range of supply, capacity and requirements.
- Heuristic crossover and non-uniform mutations

The operator generates a single offspring \( x_3 \) from two parents \( x_1 \) and \( x_2 \) according to the following rule:

\[ x_3 = \text{rounded}[r.(x_2 - x_1) + x_2] \]
where $r$ is a random number between 0 and 1, and the parent $x_2$ is no worse than $x_1$. Actually, when the generated value $x_3$ is out of bounds (upper and lower bounds) it is set equal to the corresponding violated bound. Also, note that $x_1$ and $x_2$ are “parent variables” not parents. Also $x_3$ is rounded after crossing to integer.

- Probability of crossover = 1.0, and Probability of mutation = up to 0.10
- Penalty methods for constraints handling: dynamic
- Selection for crossover: First rank all the individuals based on the objective function values. Then select randomly one from the top 10 and two from the rest. Crossover was made between the first one and better of other two.

The outputs of the algorithm were not satisfactory even for small problems with 3 sources, 2 exchange points and 3 destinations. We run the algorithm 30 times, each time for 300 generations. Most of the time, the algorithm produces too many infeasible solutions (sometimes all) and ended up either with infeasible or suboptimal (very poor feasible) solutions. To overcome this problem, we modified the algorithm to force the feasibility of offspring in each and every generation. We introduce the modified algorithm as Modified GA (MGA).

**Algorithm 2: Modified GA**
- Population size 50
- Initial population was generated randomly ensuring their feasibility.
- Heuristic crossover (as Algorithm 1) and no mutation
- Probability of crossover = 1.0
- Constraints handling: ensure feasibility with repairing mechanism
- Selection for crossover: First rank all the individuals based on the objective function values. Then select randomly one from the top 10 and two from the rest. Crossover was made between the first one and best of the other two.
- Crossover: Once the parents are selected for crossover, choose a variable randomly (for example, a path or supply amount from a source to an exchange point) and perform crossover. If the variable is within the current range (supply and capacity), we accept and reset the available supply and capacity for other variables. Otherwise, we set them at their extreme values. Once the supply is exhausted from a source, all unassigned paths from that source will be assigned to zero shipments without performing crossover operation. If any positive amount is left in any source after assigning all the paths (originating from that source) except one, then the entire amount will be allocated to that path without performing any crossover if it is less than the remaining exchange point capacity. Otherwise the minimum of available supply and exchange point capacity will be allocated.
- Ensuring Feasibility: After performing crossover following the above procedure, the problem could be infeasible in some cases because of the allocation without crossover as mentioned above and zero allocation after crossover (if the allocation is less than or equal to zero after crossover). These sorts of allocations may violate the supply constraints. Similar violation may occur in the destination side. In such a case, we apply a repair mechanism to ensure the feasibility in the final allocation.
- Repair Mechanism: Identify the sources that have supplied less than their available amounts and determine the unassigned amounts that are available for supply. Generate a list of exchange points which can receive more supplies. Randomly select an exchange point from that list and select a source with unassigned goods, and then assign to the exchange point as much as possible. Update the list of exchange points and sources, and their unassigned supply/capacity. Continue this process until all supplies are exhausted. The same procedure will be applied to the second stage (exchange point – destination) of the problem to remove infeasibility.
7.3 Computational Experience

Three instances of a small problem (with 3 sources, 2 trans-shipment points and 3 destinations) were solved and the performance compared of the above two algorithms (SGA and MGA) in term of quality of solutions. The algorithms were run for 30 times and were allowed to simulate 300 generations in each run. The results are summarized below.

<table>
<thead>
<tr>
<th>Problem Instance</th>
<th>Algorithm 1: Simple GA</th>
<th>Algorithm 2: Modified GA</th>
<th>% Difference</th>
<th>Algorithm 1: Best of 30 runs</th>
<th>Algorithm 2: Best of 30 runs</th>
<th>% Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6450</td>
<td>6450</td>
<td>0</td>
<td>16</td>
<td>30</td>
<td>46.67</td>
</tr>
<tr>
<td>2</td>
<td>11900</td>
<td>11700</td>
<td>1.71</td>
<td>3</td>
<td>23</td>
<td>86.96</td>
</tr>
<tr>
<td>3</td>
<td>38200</td>
<td>38200</td>
<td>0</td>
<td>5</td>
<td>21</td>
<td>76.19</td>
</tr>
</tbody>
</table>

From the above table, it is clear that MGA performs better for a two-stage transportation problem. All the constraints in the test problems are tight constraints. As a result, it is very hard for SGA to find a reasonable number of feasible offspring in any generation which contributes to the generation of infeasible offspring in the subsequent generation. Although we allow infeasible offspring to be parents in many situations for better convergence, it does not seem to be effective when dealing with too many tight constraints (Sarker et al., 2001a & 2001b). On the other hand, MGA ensures feasibility in every generation.

We then experimented with different sizes of test problems using the MGA algorithm. The test problems were generated by varying the number of sources, exchange points and destinations as shown in Table 3, and the capacity and demand data were generated arbitrarily random fulfilling the assumptions (i) the total supply equals total demand (balanced problem) and (ii) the total capacity of the exchange points is greater than the total supply. The unbalanced problems can be solved using the same GA either (i) by making the problem balanced with a dummy source or destination as required or (ii) by allowing a minor change in the algorithm. The number of variables and the number of constraints involved in the mathematical models of the test problems are also shown in Table 3.

<table>
<thead>
<tr>
<th>Problem Number</th>
<th>Number of sources</th>
<th>Number of Exchange points</th>
<th>Number of Destinations</th>
<th>Number of Variables</th>
<th>Number of Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>30</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>64</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>7</td>
<td>12</td>
<td>168</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>10</td>
<td>15</td>
<td>300</td>
<td>50</td>
</tr>
</tbody>
</table>

We have solved the corresponding linear programming model using a commercial optimization package LINDO/LINGO. The LP objective function value and the outputs from GA are reported in Table 4. We have experimented on a Pentium4, 1.8GHz, 256MB RAM, IBM PC. The GA ‘Overall best’ column indicates the best objective function obtained from 30 individual runs. The 30 runs average means the average of the best objective function values in 30 runs.

<table>
<thead>
<tr>
<th>Prob. No.</th>
<th>LP Objective (LINDO)</th>
<th>GA Overall best</th>
<th>GA 30 runs average</th>
<th>% Deviation (GA best with LP)</th>
<th>% Deviation (GA average with LP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11700</td>
<td>11700</td>
<td>11700</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>67900</td>
<td>67900</td>
<td>67900</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>62500</td>
<td>63000</td>
<td>63913</td>
<td>0.8</td>
<td>2.26</td>
</tr>
<tr>
<td>4</td>
<td>147200</td>
<td>147200</td>
<td>151700</td>
<td>0</td>
<td>3.06</td>
</tr>
<tr>
<td>5</td>
<td>141700</td>
<td>161800</td>
<td>170443</td>
<td>14.18</td>
<td>24.52</td>
</tr>
<tr>
<td>6</td>
<td>86750</td>
<td>112600</td>
<td>125337</td>
<td>29.80</td>
<td>44.48</td>
</tr>
<tr>
<td>7</td>
<td>91300</td>
<td>125700</td>
<td>141280</td>
<td>37.68</td>
<td>54.74</td>
</tr>
</tbody>
</table>
From columns 5 & 6 in Table 4, it is clear that MGA is providing optimal solutions for smaller problems (problem # 1 & 2) but the quality of solution is deteriorating with the increase of search space (larger problems). The solutions of the larger problems are not acceptable due to high percentage deviation from the optimal solution. It is observed that the quality of solutions is highly dependent on the quality of initial the population. It is more likely to generate a better initial population for smaller search space (smaller problems) when we use a fixed population size for all problems.

In order to improve the quality of solutions, we have conducted further experiments by changing the method of initial generation and by varying the population size. The initial population is generated as follows.

**I.** Set a list of sources with positive supply and destinations with unfulfilled demand.

**II.** Select a source or destination at random from the list, and find the cheapest route linked to transhipment points from that source / destination. Any tie will be broken arbitrarily. Assign the maximum possible amount to that route. Update the involved supply / demand and available capacity of transhipment point. If the supply is exhausted (/ demand is fulfilled), then the source (/destination) would be removed from the list.

**III.** Repeat II until all supplies are exhausted and requirements are fulfilled.

We recognize the MGA with the above change as M²GA. We run M²GA by varying the population size from 50 to 2000 as higher population size is required for better coverage of the search space in case of larger problems. As shown in Table 5, the results of M²GA are much improved although the quality of solutions is deteriorating with the increase of search space (larger problems).

<table>
<thead>
<tr>
<th>Prob. No.</th>
<th>LP Objective (LINDO)</th>
<th>GA Overall best</th>
<th>GA 30 runs average</th>
<th>% Differ. (GA best with LP)</th>
<th>% Differ. (GA average with LP)</th>
<th>GA 30 runs average (Time in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11700</td>
<td>11700</td>
<td>11700</td>
<td>0</td>
<td>0</td>
<td>0.019</td>
</tr>
<tr>
<td>2</td>
<td>67900</td>
<td>67900</td>
<td>67900</td>
<td>0</td>
<td>0</td>
<td>0.047</td>
</tr>
<tr>
<td>3</td>
<td>62500</td>
<td>62500</td>
<td>62500</td>
<td>0</td>
<td>0</td>
<td>5.82</td>
</tr>
<tr>
<td>4</td>
<td>147200</td>
<td>147200</td>
<td>147575</td>
<td>0</td>
<td>0.26</td>
<td>8.39</td>
</tr>
<tr>
<td>5</td>
<td>141700</td>
<td>142800</td>
<td>148555</td>
<td>0.7763</td>
<td>4.84</td>
<td>17.26</td>
</tr>
<tr>
<td>6</td>
<td>86750</td>
<td>87700</td>
<td>92505</td>
<td>1.0951</td>
<td>6.63</td>
<td>69.77</td>
</tr>
<tr>
<td>7</td>
<td>91300</td>
<td>94600</td>
<td>101059</td>
<td>3.6145</td>
<td>10.69</td>
<td>325.63</td>
</tr>
</tbody>
</table>

The time taken by the LP approach is about one second for problems 6 and 7. The time for problems 1 to 5 could not be collected as the package provides computational time in an increment of one second. However it is clear that LINDO/LINGO would take significantly lower computational time to solve LPs and produce global optimal solutions. In other words, the conventional LP techniques are simply too efficient compared to the other modern techniques.

We also generated a number of integer programming (IP) test problems by adding fraction (0.3 to 0.8) to some of the supply constraints right-hand-sides (supply amount). Although this modification is not practically meaningful and would not change anything in the numerical results while solving using LINDO/LINGO, it would allow us to compare the conventional integer programming (IP) techniques with GA. The time required to solve such IP test problems varies with the amount (/fraction) and place of fractions to be added. We have added some fractions randomly in different places for the above seven test problems. The times required in seconds, for the first six test problems, are 1, 6, 21, 29, 3302 and 1088 respectively. The last problem was interrupted intentionally since no feasible solution was found within 72 hours. As we can see, the computational time for GA is much lower compared to Conventional IP techniques for the test problems considered in this paper. This investigation supports the fact that GAs would do better than conventional optimization techniques for integer programming problems if we sacrifice the quality of solutions.
8. Conclusions

In this paper, we introduce evolutionary computation and analyse the similarities and differences with conventional search algorithms. The reasons for using EC in solving optimization problems and the situations where EC could be beneficial are briefly discussed. A brief review on different EAs and their components is provided. The use of EAs by solving a case problem has been demonstrated.

In this paper, we developed a GA (known as Modified GA or MGA) for a two-stage transportation problem. The problem is a standard linear programming problem. The developed GA provides better solutions than simple GA. The experimental results show that the time taken by MGA is somewhat polynomial. However the quality of solutions deteriorates with the increasing size of the problems. The LP problems were then modified to generate IP test problems. Although the LINDO/LINGO solves the IP test problems optimally, it takes much longer computational time than MGA which is an indication that GAs could be an appropriate methodology for integer programming problems.

10. References


J. Holland (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, USA.


