Chapter 9
Memetic Algorithms in Constrained Optimization

Tapabrata Ray and Ruhul Sarker

9.1 Introduction

Memetic Algorithms (MAs) are a fairly recent breed of optimization algorithms created through a synergetic coupling of global and local search strategies [615]. While predecessors of MAs, i.e. Genetic Algorithms (GAs) and Evolutionary Algorithms (EAs) have had significant success in solving a number of real life complex optimization problems in the past, their performance can be greatly improved though a hybridization with other techniques [188]. GAs or EAs hybridized with local search strategies are commonly referred as memetic algorithms. These methods are inspired by models of natural systems that combine the evolutionary adaptation of a population with individual learning within the lifetimes of its members. While, the underlying GA/EA provides the ability for exploration, the local search aids in exploitation [492]. The exploitation schemes adopted in MAs include incorporation of heuristics, approximation algorithms, local search algorithms, specialized schemes for recombination etc.

An excellent review of memetic algorithms has been presented by Ong, Lim and Chen [689]. The performance of a MA is largely dependent on the correct choice of the local search strategies (memes), identification of the sub-set undergoing local improvements and the convergence criterion used in local search strategies. In this chapter, first, we discuss constrained optimization and provide a brief review of using memetic algorithms in solving Constrained Optimization Problems (ConOPs). The representations and local search approaches used in memetic algorithms in solving different ConOPs are also described and reviewed. We also present two case studies to demonstrate the use memetic algorithms in solving ConOPs. The first case study is designed to solve constrained numerical optimization problems with traditional representation while the next is designed to solve a combinatorial

Tapabrata Ray · Ruhul Sarker
School of Engineering and Information Technology, University of New South Wales at Australian Defence Force Academy, Canberra ACT 2600, Australia
e-mail: {t.ray,r.sarker}@adfa.edu.au

optimization problem with an alternative representation. In the first case study, a local search is embedded within an evolutionary algorithm to accelerate its rate of convergence. The evolutionary algorithm unlike common EA's preserves a set of marginally infeasible solutions throughout the course of search in an attempt to identify solutions to constraint optimization problems with a higher rate of convergence. The above MA also adopts a conventional representation scheme.

In the true sprit of MAs, the second study of MA is designed to solve job shop scheduling problems through intelligent representation that includes several problem specific recombination schemes to accelerate the rate of convergence. Both case studies show the benefits of using using MAs in solving ConOPs.

9.2 Constrained Optimization

Many real-world design and decision processes require a solution to Constrained Optimization Problems (ConOPs). In general, the ConOPs can be represented mathematically as follows (without loss of generality, minimization is considered here).

$$\begin{aligned} \text{Minimize} \quad & f(\mathbf{X}) \\ \text{Subject to} \quad & g_i(\mathbf{X}) \geq 0, \quad i = 1, \ldots, m, \\ & h_j(\mathbf{X}) = 0, \quad i = 1, \ldots, p, \\ & L_i \leq x_i \leq U_i, \quad i = 1, 2, \ldots n \end{aligned} \tag{9.1}$$

where $\mathbf{X} = (x_1, \ldots, x_n)$ is a vector with n decision variables,$f(\mathbf{X})$ is the objective function, $g_i(\mathbf{X})$ is the i^{th} inequality constraint, $h_j(\mathbf{X})$ is the j^{th} equality constraint, each x_i has a lower limit L_i and an upper limit U_i.

Based on the characteristics and mathematical properties, ConOPs can be of many different types. They may contain different types of variables such as real, integer and discrete, and may have equality and/or inequality constraints. The objective and constraint functions could be either linear or nonlinear. The problem may have one or more objectives, and each objective could be either of maximization or minimization. The functions may be either continuous or discontinuous, and either unimodal or multimodal. The feasible space for such problems could be a small fraction of the search space, the entire search space or a collection of multiple disjoint spaces. The optimal solution may or may not lie on constraint boundaries. A classification of optimization problems can be found in [791]. The application of constrained optimization methods is thus wide. A few examples include: planning (resource allocation, logistics, production planning, and scheduling), engineering design (welded beam, pressure vessel, and VLSI chip design), medical science (optimization of beams for radiotherapy, DNA sequencing), and computer science (data base design and data mining).

Researchers and practitioners use both conventional mathematical optimization methods and more recent methods relying on computational intelligence to solve ConOPs. One drawback of conventional optimization methods is the fact that they require specific properties (such as convexity, continuity and differentiability) of the mathematical model and hence require simplifications of the problem via assumptions [792]. In addition, the choice of a method is determined by the problem classification and sub-classification. In contrast, algorithms based on computational intelligence are simple to implement, do not require underlying properties of the model, are amenable to parallelization and can be readily applied to a range of problems.

An EA is one such class of method based on computational intelligence where a population(set) of solutions are iteratively improved in an attempt to identify global optimal solutions. However, they usually require evaluation of numerous solutions prior to convergence resulting in higher computational times and exhibit poor convergence [455]. On the other hand, local search algorithms converge quickly to a local optimum but lack a global perspective. A combination of a population based algorithm and a local search have resulted in a new class of algorithms referred as MAs which capitalizes the benefits of both algorithms simultaneously. For example, a recent study conducted by Hasan *et al.* [378] on a job shop scheduling problem highlighted that better quality of solutions could be obtained using MA with reduced computational effort as compared to genetic algorithms. Boudia and Prins [79] indicated that the solutions produced by memetic algorithms, for an integrated production-distribution problem, made significant savings as compared to others. More recently Singh *et al.* [816] reported the results of their infeasibility empowered memetic algorithm on a set of CEC-2010 constrained optimization benchmarks. It is also important to highlight that MAs are also attractive for dynamic optimization problems where an improved rate of convergence is required along with the ability to search for global optima. Isaacs *et al.* [407] have reported the performance of a memetic algorithm on dynamic bi-objective problems highlighting the benefits over evolutionary algorithms.

While population based methods such as EAs perform well as compared to conventional methods on unconstrained optimization problems, their performance on constrained optimization problems is not exceptionally good. Common search operators of EAs (such as crossover and mutation) are blind to the constraints. As a consequence, the candidate solutions generated by these operators may violate constraints [126]. Hence, mechanisms for constraint handling play an important role on the performance of such algorithms. Over the past decade, various constraint handling techniques have been proposed in the context of evolutionary optimization [126, 133, 195, 597, 908]. These techniques can be grouped as: penalty functions, special representations and operators, repair algorithms, separation of objectives and constraints, and hybrid methods. The purpose of these methods is to find the constraint violations, and use such information to rank and select the individuals for reproduction. Such methods are referred as MAs with conventional representation and are discussed in depth in the following section.

While many MAs adopt conventional representation i.e. the solution represented as a vector of decision variables, there are many which focus on the underlying solution representation scheme and include specialized representation and/or repair methods to deal with constraints efficiently. The details of such methods are discussed under the broad context of MAs with alternative representation. Two case studies are carefully selected to illustrate the behavior of both these classes of MAs.

9.3 Classification of MAs

As observed in the literature, the trend of MAs used for constrained optimization can be represented by the classification shown in Figure 9.1.

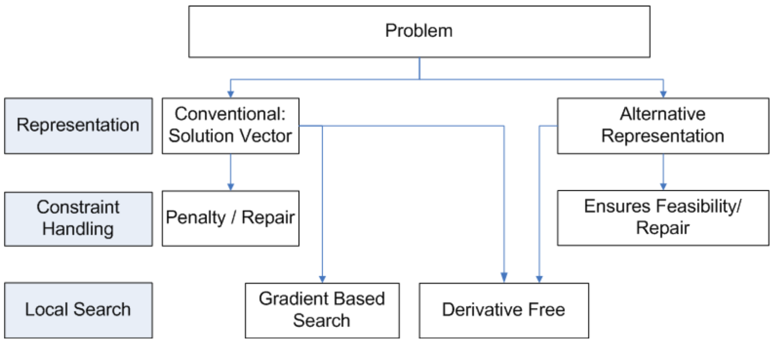

Fig. 9.1. Classification of MAs

Some examples of MAs based on the above classification are given in Table 9.1. It is interesting to observe that MAs, with chromosome representation based on solution vector, use penalty or repair method for dealing with constraint violation. On the other hand, MAs, with alternative chromosome representation, use derivative free local search method, and use either feasible individuals or repair infeasible individuals to deal with constraints. From the review in an earlier section, it is clear that the alternative representation is popular for solving combinatorial optimization problems.

9.4 MAs with Conventional Representation

In this section, we will discuss optimization problem solving, using MAs, where the complete mathematical model is available and a chromosome is represented as a vector of decision variables.

Handoko *et al.* [356] developed a MA where a GA was combined with a gradient based local search to solve nonlinear programming problems. The constraint violation was handled using three simple rules as of Deb [195] : (i) the feasible individual is preferred over the infeasible one; (ii) for two feasible individuals, the individual with better fitness is preferred; and (iii) for two infeasible individuals, the individual with lower constraint violation is preferred. Their experimental results indicated that MAs outperformed conventional algorithms in terms of both quality of solution and the rate of convergence.

Table 9.1. Examples of MAs in literature

	Representation (as discussed earlier)		
	Based on solution vector	Alternative Representation	
	Constraint handling	Constraint handling	
	Penalty/Repair	Penalty/Repair	Ensures feasibility
Gradient based local search	Handoko *et al.* [356] Singh *et al.* [816] Kelner *et al.* [455] Barkat Ullah *et al.* [44]		
Derivative free local search	Lin and Liang [518] Barkar Ullah *et al.* [45] Boudia and Prins [79] Park *et al.* [713]	Hasan *et al.* [377, 378]	Prins [734, 735] Fallahi *et al.* [249] Ngueveu *et al.* [662] Mendoza *et al.* [579] Marinakis & Marinaki [555]

Singh *et al.* [816] designed an infeasibility empowered MA for solving constrained optimization problems where an underlying EA was combined with a local search (Sequential Quadratic Programming (SQP)). The constraint violation was tackled using principles of infeasible solution embedding Singh *et al.* [759] and the results were reported for the series of 18 constrained test problems as introduced in CEC-2010 competition.

Lin and Liang [518] proposed a hybrid algorithm where a GA was combined with an adaptive penalty method and a line search technique (Hooke and Jeeves). The performance of the algorithm on a series of 13 well-known benchmark problems established its robustness.

Kelner *et al.* [455] proposed a hybrid algorithm as a combination of a GA and a local search strategy based on the interior point method, for solving constrained multi-objective mathematical models. The constraints were handled using the rules proposed by Deb [195]. The efficiency of the algorithm was demonstrated using a number of test problems.

Barkat Ullah *et al.* [44] proposed an agent based memetic algorithm in which four local search algorithms were used for adaptive learning. The algorithms included random perturbation, neighborhood and gradient search methods. Subsequently, another specialized local search method was designed to deal with equality constraints (Barkat Ullah *et al.* [45]. The constraints were handled using the rules proposed by Deb [195]. Although the algorithm identified high quality solutions on the set of 13 benchmarks, the computational time was a bit longer than state-of-the-art algorithms (Runarsson and Yao [782]) as the underlying lattice-like environment and orthogonal crossovers consumed a fair amount of time.

Liu *et al.* [527] developed a memetic co-evolutionary differential evolution algorithm where the population was divided into two sub-populations. The purpose of one sub-population is to minimize the fitness function, and the other is to minimize the constraint violation. The optimization was achieved through interactions between the two sub-populations. No penalty coefficient was used in the method while a Gaussian random number was used to modify the individuals when the best solution remained unchanged over several generations. The results indicate the algorithm being computationally inexpensive in terms of memory requirements and CPU times and efficient when compared with existing state of the art algorithms.

While most of the applications reported above are tested on mathematical benchmarks, several practical applications have also adopted conventional representation. Boudia and Prins [79], Park *et al.* [713], and Berretta and Rodrigues [61] dealt with three different practical problems and in all studies chromosomes were designed using conventional representation. Boudia and Prins [79] considered the problem of cost minimization of a production-distribution system. The moves (local search) used were 2-OPT, relocate a customer, and swap between two customers. A repair mechanism was also applied for constraint satisfaction. The algorithm reported significant savings as compared to two other existing methods. Park *et al.* [713] combined a GA with a tunnel-based dynamic programming scheme (as a local search) to solve highly constrained non-linear discrete dynamic optimization problems arising from long-term planning. The infeasible solutions were repaired by regenerating partial characters. The algorithm successfully solved reasonable sized practical problems which cannot be solved using conventional approaches. A multi-stage capacitated lot-sizing problem was solved by the memetic algorithm proposed by Berretta and Rodrigues [61] using heuristics as local search coupled with usual crossover and mutation operators. The results using the above method were better than those generated using existing heuristics.

9.5 MAs with Alternative Representations

While the above section highlighted a number of successful MAs that have been designed to solve constrained optimization problems using conventional representation schemes, there are also a number of MAs that have been designed to solve problems using alternative representation schemes. Combinatorial problems require many integer (mainly binary) variables and logical constraints to represent them mathematically. Hence, a chromosome design based on the decision variables of the mathematical model as a vector becomes too long. Just to give an idea, let us consider a single variable piecewise linear function or a continuous nonlinear function that can be approximated by a number of piecewise linear functions. To express these functions mathematically for n segments, we need (n+1) real variables, (n-1) binary variables and (n+1) logical constraints. So, $2n$ variables in the chromosome and additional (n + 1) constraints are required to represent the function of a single variable. In alternative chromosome design, one can use just one variable as illustrated in Ray and Sarker [758]. The applications of alternative representations in MAs are briefly reviewed below.

Prins [734] developed a memetic algorithm for solving vehicle routing problems (VRPs) which outperformed most Tabu Search (TS) heuristics (best known algorithms for VRPs at that time) on a number of test instances. The solution was represented using a TSP-like permutation chromosome, without trip delimiters, and local search procedures (like moving or swapping some nodes) were used in lieu of mutation for search. Later, Prins [735] proposed two more memetic algorithms for heterogeneous fleet vehicle routing problems (HFVRPs) that are based on chromosome encoded as giant tours, without trip delimiters. Such chromosomes do not directly represent the decision variables of the corresponding mathematical model of the problem. In both of the above studies, Prins applied an optimal evaluation procedure that splits the tours into feasible trips and assign vehicles to them. As a result, no repair mechanism or penalty method was required. The perturbation was achieved through the relocation of one customer, the exchange of two customers, and 2-OPT moves operated on one or two selected routes. In order to maintain diversity, a distance measure in the solution space was used. The algorithm is one of the most successful algorithms for vehicle fleet mix problem with both fixed and variable costs (VFMP-FV) that has been able to discover six new best solutions to benchmark problems.

El Fallahi *et al.* [249] and Ngueveu *et al.* [662] developed a memetic algorithm for multi-compartment vehicle routing problems (MC-VRPs) and cumulative vehicle routing problems (CCVRPs) respectively. In these algorithms, the chromosome representation and evaluation procedure are similar to Prins [734]. However, the moves (local search) in the first algorithm are based on 2-OPT, relocate and I-interchange and the second include 2-OPT, relocation of one customer and exchange of two customers. Mendoza *et al.* [579] proposed a memetic algorithm for a variant of MC-VRPs with a different representation known as the genetic vehicle representation (GVR). In GVR, each permutation contains an ordered set of customers representing a route. This representation allows the straightforward application of the selected crossover, mutation and local search operators designed to work on independent routes. The authors used relocate and 2-OPT as the local search schemes.

Marinakis and Marinaki [555] proposed a memetic algorithm for the solution of VRPs. The MA makes use of a GA framework with an expanding neighborhood search. Although, significantly better solutions were reported on two sets of benchmark instances, there is no comparison on computational time.

Hasan *et al.* [377, 378] developed a memetic algorithm for solving job-shop scheduling problems. They used job pair-relation based genotype representation, priority rules as local search, and a repair mechanism for changing the infeasible individuals into feasible. It is generally accepted that the time taken per generation of MA would be higher than that of GA. However Hasan *et al.* [378] proved that MA, as compared to GA, not only improves the quality of solutions but also reduces the overall computational time. The proposed MA improved the average of the best solutions over GA by 2.623%, while reducing the computational time by 40.57% on average per problem. It is also important to take note that these are based on 40 well-known series of benchmark problems.

9.6 Numerical Case Studies

Two case studies are discussed in depth in the following sub-sections.

9.6.1 Case Study 1: Infeasibility Empowered Memetic Algorithm for Constrained Optimization Problems: MA with Conventional Representation

In this section we present an Infeasibility Empowered Memetic Algorithm (IEMA) which is a combination of Infeasibility Driven Evolutionary Algorithm(IDEA) and a local search based on Sequential Quadratic Program (SQP). IDEA is a derived variant of EAs in which a small proportion of marginally infeasible solutions are preserved to accelerate the rate of convergence. While most EAs rank feasible solutions above infeasible solutions, IDEA ranks solutions based on the original objectives along with additional objective representing constraint violation measure. In addition, "good" infeasible solutions are ranked higher than the feasible solutions, and thereby the search proceeds through both feasible and infeasible regions, resulting in greater rate of convergence to optimal solutions. The studies reported in [759, 817] indicate that IDEA has better rate of convergence over conventional EAs for a number of constrained single and multi-objective optimization problems. The following subsections provide the background of IDEA and necessary details of IEMA.

9.6.1.1 Infeasibility Driven Evolutionary Algorithm (IDEA)

A generalized single-objective optimization problem can be formulated as shown in (9.1). It is a usual practice to convert the equality constraints to inequality constraints using a small tolerance (*i.e.* $h(\mathbf{x}) = 0$ is converted to $|h(\mathbf{x})| \leq \varepsilon$). Hence, the discussion presented here is with regards to presence of inequality constraints only.

To effectively search the design space (including the feasible and the infeasible regions near constraint boundaries), the original single objective constrained optimization problem is reformulated as bi-objective unconstrained optimization problem as shown in (9.2).

$$\begin{aligned} \text{Minimize} \quad f_1'(\mathbf{x}) &= f_1(\mathbf{x}) \\ f_2'(\mathbf{x}) &= \text{violation measure} \end{aligned} \tag{9.2}$$

The additional objective represents a measure of constraint violation, which is referred to as "violation measure". It is based on the amount of relative constraint violations among the population members. Each solution in the population is assigned m ranks, corresponding to each m constraints. The ranks are calculated as follows. To get the ranks corresponding to i^{th} constraint, all the solutions are sorted based on the constraint violation value of i^{th} constraint. Solutions that do not violate the constraint are assigned rank 0. The solution with the least constraint violation value gets rank 1, and the rest of the solutions are assigned increasing ranks in the

Algorithm 19. Infeasibility Driven Evolutionary Algorithm (IDEA)

```
1  begin
       // Given population size $N$ number of generations $N_G > 1$
          and Proportion of infeasible solutions $0 < \alpha < 1$
2      $N_{inf} \leftarrow \alpha * N$;
3      $N_f \leftarrow N - N_{inf}$;
4      set $pop_1 \leftarrow$ Initialize();
5      Evaluate($pop_1$);
6      for $i = 2$ to $N_G$ do
7          $childpop_{i-1} \leftarrow$ Evolve($pop_{i-1}$);
8          Evaluate($childpop_{i-1}$);
9          $(S_f, S_{inf}) \leftarrow$ Split($pop_{i-1} + childpop_{i-1}$);
10         Rank($S_f$);
11         Rank($S_{inf}$);
12         $pop_i \leftarrow S_{inf}(1 : N_{inf}) + S_f(1 : N_f)$
13     endfor
14 end
```

ascending order of their constraint violation values. The process is repeated for all the constraints and as a result each solution in the population gets assigned m ranks. The violation measure is the sum of these m ranks corresponding to m constraints.

The main steps of IDEA are outlined in Algorithm 19. IDEA uses simulated binary crossover (SBX) and polynomial mutation operators to generate offspring from a pair of parents selected using binary tournament as in NSGA-II [200]. Individual solutions in the population are evaluated using the original problem definition (9.1) and the infeasible solutions are identified. The solutions in the parent and offspring population are divided into a feasible set (S_f) and an infeasible set (S_{inf}). The solutions in the feasible set and the infeasible set are ranked separately using the non-dominated sorting and crowding distance sorting [200] based on 2 objectives as per the modified problem definition (9.2). The solutions for the next generation are selected from both the sets to maintain infeasible solutions in the population. In addition, the infeasible solutions are ranked higher than the feasible solutions to provide a selection pressure to create *better* infeasible solutions resulting in an active search through the infeasible search space.

A user-defined parameter α is used to maintain a set of infeasible solutions as a fraction of the size of the population. The numbers N_f and N_{inf} denote the number of feasible and infeasible solutions as determined by parameter α. If the infeasible set S_{inf} has more than N_{inf} solutions, then first N_{inf} solutions are selected based on their rank, else all the solutions from S_{inf} are selected. The rest of the solutions are selected from the feasible set S_f, provided there are at least N_f number of feasible solutions. If S_f has fewer solutions, all the feasible solutions are selected and the rest are filled with infeasible solutions from S_{inf}. The solutions are ranked from 1 to N in the order they are selected. Hence, the infeasible solutions selected first will be ranked higher than the feasible solutions selected later.

Algorithm 20. Infeasibility Empowered Memetic Algorithm (IEMA)

```
1  begin
       // Given population size $N$ number of generations $N_G > 1$
          and Proportion of infeasible solutions $0 < \alpha < 1$
2      $N_{inf} \leftarrow \alpha * N$;
3      $N_f \leftarrow N - N_{inf}$;
4      $pop_1$ = Initialize();
5      Evaluate($pop_1$);
6      for $i = 2$ to $N_G$ do
7          $childpop_{i-1} \leftarrow$ Evolve($pop_{i-1}$);
8          Evaluate($childpop_{i-1}$);
9          $(S_f, S_{inf}) \leftarrow$ Split($pop_{i-1} + childpop_{i-1}$);
10         Rank($S_f$);
11         Rank($S_{inf}$);
12         $pop_i \leftarrow S_{inf}(1 : N_{inf}) + S_f(1 : N_f)$;
13         $\mathbf{x} \leftarrow$ Random solution in $pop_i$;
14         $\mathbf{x_{best}} \leftarrow$ Local_ search ($\mathbf{x}$);
           // $\mathbf{x_{best}}$ is the best solution found using local search
              from $\mathbf{x}$
15         Replace worst solution in $pop_i$ with $\mathbf{x_{best}}$;
16         Rank($pop_i$);
17         Rank the solutions again in $pop_i$
18     endfor
19 end
```

9.6.1.2 Infeasibility Empowered Memetic Algorithm (IEMA)

The proposed algorithm IEMA is constructed using IDEA as the baseline algorithm. For single objective problems, a local search can be a very efficient tool for optimization. However, its performance is largely dependent on the starting solution. The proposed algorithm tries to exploit the advantages of both these approaches, i.e. 1) searching near the constraint boundaries by preserving marginally infeasible solutions during the search, and 2) the effectiveness of local search to expedite the convergence in potentially optimal regions of the search space. Hence, we refer to the proposed algorithm as Infeasibility Empowered Memetic Algorithm (IEMA).

The proposed IEMA is outlined in algorithm 20. In IEMA, during each generation, apart from the evolution of the solutions in IDEA, a local search is done from a random solution in the population, for a prescribed number of function evaluations (set to 2000 here). Sequential Quadratic Programming (SQP) [729] has been used in the presented studies for the local search. Thereafter, the worst solution in the population is replaced by the best solution found from the local search. The ranking of solutions is done in the same way as done in IDEA. The injection of good quality solutions found using the local search guides the population towards potentially optimal regions of the search space. The evolved solutions in turn act as good starting solutions for the local search in subsequent generations.

9.6.1.3 Results on CEC-2010 Benchmark Problems

- **Experimental setup:** The performance of IEMA is presented for one of the most recent difficult set of constrained optimization benchmarks, i.e. that of IEEE CEC-2010, constrained optimization competition. Twenty five runs of the proposed algorithm IEMA are done on each of the test problems C01 - C18 [550]. The parameters used for IEMA are same for each problem, i.e. no tuning of parameters is done across the problems. The parameters are listed in Table 9.2. A maximum of 2000 function evaluations are allotted to the local search within each generation.

Table 9.2. Parameters used for IEMA

Parameter	Value
Population Size	200
Max. FES	for 10D problems: 200000
	for 30D problems: 600000
Crossover Probability	0.9
Crossover index	15
Mutation Probability	0.1
Mutation index	20
Infeasibility Ratio (α)	0.9

- **PC configuration:** All the runs are made on a cluster with the compute nodes DL140G3 5110 NHP Sata, with following configuration:

  1. Processor - Dual-core Intel Xeon 5110
  2. RAM - 4GB
  3. Operating system - Redhat Linux

  IEMA algorithm is implemented in Matlab 2008a.
- **Summary of results:** The results for 10D problems are shown in Table 9.3, whereas the results for 30D problems are listed in Table 9.4. To determine the median, following procedure is adopted. All the runs in which a feasible solution was found are sorted based on the best function value obtained. Thereafter, all the runs in which no feasible solutions are found are sorted based on the mean constraint violation of the best (infeasible) solution found. Feasible runs are ranked above infeasible runs. In the sorted list, the 13^{th} solution is reported as the median solution (only if the median is feasible). The best, mean and worst runs reported in the tables are based only on the runs in which at least one feasible solution was found. The number of such feasible runs are also reported in the tables for each problem. The median value, if infeasible is also not reported.

From Table 9.3, it is observed that for 10D problems, IEMA is able to achieve all (25) feasible runs for 12 problems out of 18. The best value obtained for many problems are much better than the median and worst values, indicating a possibility of highly multimodal objective functions. This also results in a correspondingly high value of standard deviation (std), as seen from the table.

For 30D problems (Table 9.4), the results are worse as compared to the 10D problems. For 4 out of 18 functions, no feasible solution was identified. Among the remaining 14 functions, all 25 runs were feasible for 11 problems. Once again, the results are seen to have a high standard deviation value as in 10D case, and the best values found are much better than the median/worse values for some of the problems.

Table 9.3. Performance of IEMA on 10D problems

	C01	C02	C03	C04	C05	C06
Best	-0.74731	-2.27771	1.46667e-16	-9.98606e-06	-483.611	-578.662
Median	-0.74615	-2.27771	3.2005e-15	-9.95109e-06	-483.611	-578.662
Mean	-0.743189	-2.27771	6.23456e-07	-9.91135e-06	-379.156	-551.47
std	0.00433099	1.82278e-07	1.40239e-06	8.99217e-08	179.424	73.5817
Feasible	25	25	25	25	24	24
	C07	C08	C09	C10	C11	C12
Best	1.74726e-10	1.00753e-10	1.20218e-09	5.4012e-09	-0.00152271	-10.9735
Median	1.9587e-09	3.94831e-09	333.32	42130.4	-0.00152271	-0.199246
Mean	3.25685e-09	4.0702	1.95109e+12	2.5613e+12	-0.00152271	-0.648172
std	3.38717e-09	6.38287	5.40139e+12	3.96979e+12	2.73127e-08	2.19928
Feasible	25	25	23	19	24	24
	C13	C14	C15	C16	C17	C18
Best	-68.4294	8.03508e-10	9.35405e-10	4.44089e-16	9.47971e-15	2.23664e-15
Median	-68.4294	1.29625e-08	26.1715	0.0320248	2.59284e-12	6.78077e-15
Mean	-68.0182	56.3081	1.57531e+08	0.0330299	0.00315093	1.61789e-14
std	1.40069	182.866	6.04477e+08	0.0226013	0.0157547	3.82034e-14
Feasible	25	25	25	25	25	25

- **Convergence plots:** The convergence plots for C09, C10, C14, C15, C17 and C18 are shown in Figure 9.2. The plots show the feasible solutions only, for the best runs corresponding to these problems. The objective values have been plotted in log scale in order to aid visualization.
- **Time complexity:** The time complexity of the algorithm is shown in Table 9.5. $T1$ and $T2$ are as defined in [550]. $T1$ represents the average (across C01-C18) time taken for evaluating the problem 10000 times, whereas $T2$ represents the average time taken across C01-C18 by the algorithm IEMA to run through 10000 FES.

Table 9.4. Performance of IEMA on 30D problems

	C01	C02	C03	C04	C05	C06
Best	-0.821883	-2.28091	-	-	-286.678	-529.593
Median	-0.819145	-2.27767	-	-	-	-
Mean	-0.817769	-1.50449	-	-	-270.93	-132.876
std	0.00478853	2.14056	-	-	14.1169	561.042
Feasible	25	25	0	0	4	2
	C07	**C08**	**C09**	**C10**	**C11**	**C12**
Best	4.81578e-10	1.12009e-09	7314.23	27682	-	-
Median	6.32192e-10	0.101033	7.91089e+06	1.1134e+07	-	-
Mean	8.48609e-10	17.7033	2.98793e+07	1.58342e+07	-	-
std	4.84296e-10	40.8025	4.50013e+07	1.68363e+07	-	-
Feasible	25	25	25	25	0	0
	C13	**C14**	**C15**	**C16**	**C17**	**C18**
Best	-68.4294	3.28834e-09	31187.6	6.15674e-12	9.27664e-10	1.37537e-14
Median	-67.6537	7.38087e-09	7.28118e+07	1.26779e-10	5.67557e-06	2.12239e-14
Mean	-67.4872	0.0615242	2.29491e+08	0.00163294	0.0883974	4.73841e-14
std	0.983662	0.307356	4.64046e+08	0.0081647	0.15109	6.5735e-14
Feasible	25	25	25	25	25	25

Table 9.5. Time complexity of IEMA (in seconds)

	$T1$	$T2$	$(T2-T1)/T1$
10D problems	2.57636	9.05104	2.51312
30D problem	2.57854	13.2825	4.1512

9.6.2 Case Study 2: MA with Alternative Representation

The job-shop scheduling problem (JSSP) is a well-known practical planning problem in the manufacturing sector. A classical JSSP is a combination of N jobs and M machines. Each job consists of a set of operations that has to be processed, on a set of known machines, and where each operation has a known processing time. A schedule is a complete set of operations, required by a job, to be performed on different machines, in a given order. In addition, the process may need to satisfy other constraints such as (i) no more than one operation of any job can be executed simultaneously and (ii) no machine can process more than one operation at the same time. The objectives usually considered in JSSPs are the minimization of makespan. The total time between the starting of the first operation and the ending of the last operation, is termed as the "makespan". We first develop a traditional GA for solving JSSPs. We then proposed three versions of memetic algorithms using three new

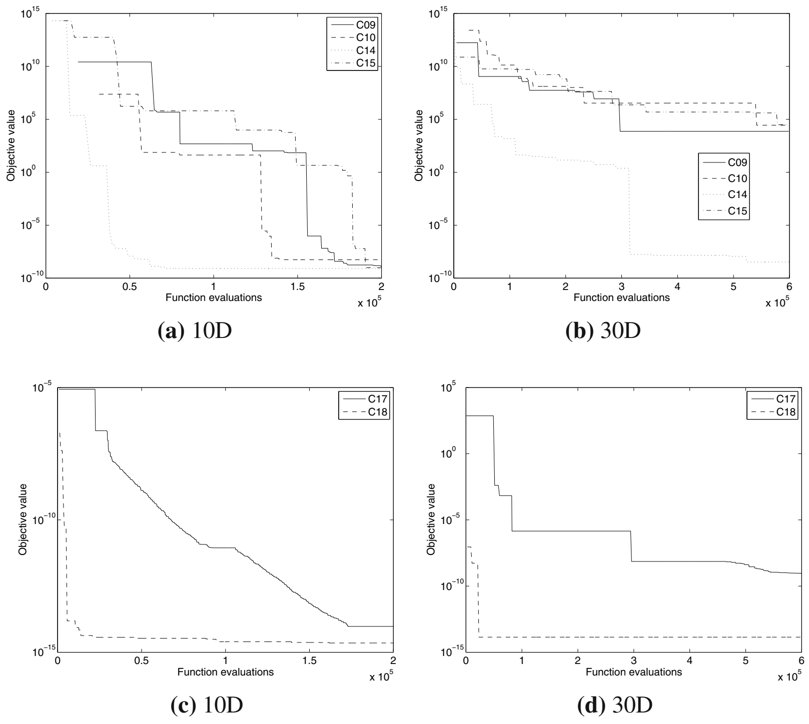

Fig. 9.2. Convergence plots (y-axis is in log scale)

priority rules for improving the performance of traditional GA, namely: partial re-ordering (PR), gap reduction (GR) and restricted swapping (RS). The performances of our proposed algorithms are analyzed by solving 40 well-known benchmark problems. The chromosome representation, priority rules and the performance analysis are briefly discussed below.

9.6.2.1 Chromosome Representation

In this study, we do not solve the mathematical model of the job shop problem. Instead we develop GA and MA for solving the problem directly. We select the job pair-relationship based representation for the genotype, as in [649, 946], due to the flexibility of applying genetic operators to it. In this representation, a chromosome is symbolized by a binary string, where each bit stands for the order of a job pair (u,v) for a particular machine m. This binary string acts as the genotype of individuals. The corresponding phenotype represents the job sequence for each machine. Further details on the chromosome design can be found in Hasan *et al.* [377].

9.6.2.2 Priority Rules

The priority rules developed for this study are as follows.

- **Partial Reordering (PR):** In this rule, we identify the machine which is the deciding factor for the makespan and the last job (say J^*) that is to be processed by that machine. That machine can be termed as the bottleneck machine in the chromosome under consideration. Then we find the machine (say M^*) that is required by the first operation of the identified job J^*. The re-ordering rule then suggests that the first operation of the identified job (J^*) must be the first task on machine M^* if it is not already scheduled. If we move the job J^* from its current position to the 1^{st} position, we may need to push some other jobs currently scheduled on machine M^* to the right. In addition, it may provide an opportunity to shift some jobs to the left on other machines. The overall process helps to reduce the makespan for some chromosomes.
- **Gap Reduction (GR):** After each generation, the generated phenotype usually leaves some gaps between the jobs. Sometimes, these gaps are necessary to satisfy the precedence constraints. However, in some cases, a gap could be removed or reduced by placing a job from the right side of the gap. For a given machine, this is like swapping between a gap from the left and a job from the right of a schedule. In addition, a gap may be removed or reduced by simply moving a job to its adjacent gap at the left. This process would help to develop a compact schedule from the left and continuing up to the last job for each machine. Of course, it must ensure no conflict or infeasibility before accepting the move.
- **Restricted swapping (RS):** For a given machine, the restricted swapping rule allows swapping between the adjacent jobs if and only if the resulting schedule is feasible. This process is carried out only for the job which takes the longest time for completion.

9.6.2.3 Implementation

First, we implement a simple GA for solving JSSPs. We use simple two point crossover and bit flip mutation as reproduction operators. We then implemented three versions of MAs by introducing the priority rules as local search techniques as follows:

- MA(PR): Partial re-ordering rule with GA,
- MA(GR): Gap reduction rule with GA, and
- MA(GR-RS): Gap reduction and restricted swapping rule with GA

In both GA and MA, we apply elitism in each generation to preserve the best solution found so far, and also to inherit the elite individuals more than the rest. In performing the crossover operation, we use the tournament selection that chooses one individual from the elite class of the individuals (*i.e.* the top 15%) and two individuals from the rest. This selection then plays a tournament between the last two and performs crossover between the winner and the elite individual. We rank the individuals on the basis of the fitness value. From our extensive parametric analysis, we have chosen the crossover and mutation rate as 0.45 and 0.35 respectively. We set the population size to 2500 and the number of generations to 1000. Note that JSSPs usually require a higher population size. For example, Pezzella *et al.* [721] used a population size of 5000 even for 10×10 problems. In our approach, GR is applied to every individual. On the other hand, we apply PR and RS to only 5% of randomly selected individuals in every generation. To test the performance of our proposed algorithms, we have solved the 40 benchmark problems designed by Lawrence [509] and have compared the results.

9.6.2.4 Result and Analysis

Each problem was run 30 times and Table 9.6 compares the performance of four algorithms we implement [GA, MA(PR), MA(GR), and MA(GR-RS)] in terms of the % average relative deviation (ARD) from the best result published in the literature, the standard deviation of % relative deviation (SDRD), and the average number of fitness evaluations required. From Table 9.6, it is clear that the performance of the MAs are better than the GA, and MA(GR) is better than both MA(PR) and GA. The addition of RS to MA(GR), which is known as MA(GR-RS), has clearly enhanced the performance of the algorithm. Out of the 40 test problems, both MA(GR) and MA(GR-RS) obtained exact optimal solutions for 23 problems. In addition, MA(GR-RS) obtained optimal solutions for another 4 problems and substantially improved solutions for 10 other problems. In general, these two algorithms converged quickly, which can be seen from the average number of fitness evaluations.

Table 9.6. Comparing our four algorithms for 40 test problems

Algorithm	Optimal Found	ARD (%)	SDRD	Average # of generations	Average # of Fitness eval.(10^3)	Average Computational time (s)
GA	15	3.591	4.165	270.93	664.90	201.60
MA(PR)	16	3.503	4.192	272.79	660.86	213.42
MA(GR)	23	1.360	2.250	136.54	356.41	105.87
MA(GR-RS)	27	0.968	1.656	146.63	388.58	119.81

As shown in Table 9.6, the addition of the local search techniques to GA (for the last two MAs) not only improves the quality of solutions significantly but also helps in converging to the solutions with a lower number of generations and a lower total number of fitness evaluations. However, as the local search techniques require additional computation, the computational time per generation for all three MAs is higher than GA. For example, the average computational time taken per generation by the algorithms GA, MA(PR), MA(GR) and MA(GR-RS) are 0.744, 0.782, 0.775 and 0.817 seconds respectively. Interestingly, the overall average computational time per test problem solved, for the algorithm MA(GR-RS), is the lowest among the four algorithms implemented. As of Table 9.6, for all 40 test problems, the algorithm MA(GR-RS) improved the average of the best solutions over GA by 2.623%, while reducing the computational time by 40.57% on average per problem. This clearly demonstrates the strength of MAs.

9.7 Summary and Conclusions

This chapter provides a review of various memetic algorithms that have been proposed over the years to deal with constrained optimization problems. Details of two distinct and widely different classes of MAs are presented in the chapter. The first MA adopts a conventional representation scheme and employs a population based global search and a SQP for local search. The population based global search component of MA explicitly maintains a fraction of marginally infeasible solutions in a quest to accelerate its rate of convergence. The second MA and its variants on the other hand is designed to efficiently solve job shop scheduling problems. The algorithm employs specialized representation, recombination and local search strategies/heuristics in an attempt to improve the rate of convergence. The examples clearly highlight the potential benefits that can be realized through the use of MAs and the range of local learning schemes that can be used to further enhance its performance.

Acknowledgements. The authors would like to acknowledge the help received from Dr. Kamrul Hasan for providing results for the Job Shop Scheduling problem, and Mr. Hemant Kumar Singh for benchmarking IEMA and formatting of this manuscript. The efforts of Dr. Amitay Isaacs to generate several parts of the code is also acknowledged.