Appl Intell (2007) 26:183–195
DOI 10.1007/s10489-006-0018-y

Genetic operators for combinatorial optimization in TSP and microarray gene ordering

Shubhra Sankar Ray · Sanghamitra Bandyopadhyay · Sankar K. Pal

Published online: 9 November 2006

Abstract This paper deals with some new operators of genetic algorithms and[-27pc] demonstrates their effectiveness to the traveling salesman problem (TSP) and microarray gene ordering. The new operators developed are nearest fragment operator based on the concept of nearest neighbor heuristic, and a modified version of order crossover operator. While these result in faster convergence of Genetic Algorithm (GAs) in finding the optimal order of genes in microarray and cities in TSP, the nearest fragment operator can augment the search space quickly and thus obtain much better results compared to other heuristics. Appropriate number of fragments for the nearest fragment operator and appropriate substring length in terms of the number of cities/genes for the modified order crossover operator are determined systematically. Gene order provided by the proposed method is seen to be superior to other related methods based on GAs, neural networks and clustering in terms of biological scores computed using categorization of the genes.

Keywords Microarray · Gene analysis · Data mining · Biocomputing · Evolutionary algorithm · Soft computing

S. S. Ray (✉) · S. K. Pal
Center for Soft Computing Research: A National Facility, Indian Statistical Institute, Kolkata 700108, India
e-mail: shubhra_r@isical.ac.in

S. K. Pal
e-mail: sankar@isical.ac.in

S. Bandyopadhyay
Machine Intelligence Unit, Indian Statistical Institute, Kolkata 700108, India
e-mail: sanghami@isical.ac.in

1 Introduction

The Traveling Salesman Problem (TSP) is one of the top ten problems, which has been addressed extensively by mathematicians and computer scientists. It has been used as one of the most important test-beds for new combinatorial optimization methods [1]. Its importance stems from the fact there is a plethora of fields in which it finds applications e.g., shop floor control (scheduling), distribution of goods and services (vehicle routing), product design (VLSI layout), microarray gene ordering and DNA fragment assembly. Since the TSP has proved to belong to the class of NP-hard problems [2], heuristics and metaheuristics occupy an important place in the methods so far developed to provide practical solutions for large instances and any problem belonging to the NP-class can be formulated with TSP. The classical formulation is stated as: Given a finite set of cities and the cost of traveling from city I to city j, if a traveling salesman were to visit each city exactly once and then return to the home city, which tour would incur the minimum cost?

Over decades, researchers have suggested a multitude of heuristic algorithms, such as genetic algorithms (GAs) [3–6], tabu search [7, 8], neural networks [9, 10], and ant colonies [11] for solving TSP. Of particular interest are the GAs, due to the effectiveness achieved by this class of techniques in finding near optimal solutions in short computational time for large combinatorial optimization problems. The state-of-the-art techniques for solving TSP with GA incorporates various local search heuristics including modified versions of Lin-Kernighan (LK) heuristic [12–15]. It has been found that, hybridization of local search heuristics with GA for solving TSP leads to better performance, in general. Some important considerations in integrating GAs and Lin-Kernighan heuristic, selection of a proper representation strategy, creation of the initial population and designing of various genetic

operators are discussed in [16]. A comprehensive discussion regarding different representation strategies for TSP is provided in [1].

For creating the initial population, random population based approach and nearest neighbor tour construction heuristic (NN) approach are commonly used. Regarding the random population based approach, consider the investigations in [5] and [6] as examples. A GA with immunity (IGA) is developed in [5]. It is based on the theory of immunity in biology, which mainly constructs an immune operator accomplished in two steps: (a) a vaccination and (b) an immune selection. Strategies and methods of selecting vaccines and constructing an immune operator are also mentioned in [5]. IGA can improve the searching ability and adaptability of TSP. Two operators of GA, namely, knowledge based multiple inversion (KBMI) and knowledge based neighborhood swapping (KBNS) are reported in [6]. KBMI helps in exploring the search space efficiently and prevents the GA from getting stuck in the local optima, whereas, KBNS, a deterministic operator, helps the stochastic environment of the working of the GA to derive an extra boost in the positive direction. The corresponding GA for solving TSP is referred to as SWAP_GATSP [6].

Nearest neighbor (NN) tour construction heuristic is a common choice to construct the initial population of chromosome for solving TSP with GAs. Investigations in this line include [4, 17–19]. In [4] a modified multiple-searching genetic algorithm (MMGA) is used with two kinds of chromosomes (namely, conservative and explorer). These two chromosomes operate under different crossover and mutation rates for tour improvement and to avoid the possibility of being trapped at local optima in TSP. Since the NN heuristic takes a locally greedy decision at each step, it is found that several cities that are neglected earlier, may need to be inserted at high costs in the end. This leads to severe mistakes in path construction.

Crossover operators of GAs are seen to rectify the mistakes in path construction by NN or any other approach. Various crossover operators such as order crossover [20], cycle crossover [21], partially matched crossover [3], edge-recombination crossover [22, 23], and matrix crossover [24] have been suggested for the TSP. Order crossover has been observed to be one of the best in terms of quality and speed, and yet is simple to implement for solving TSP using GA [1, 3, 6]. However, the random length of substring, chosen from the parent string for performing crossover may increase the computational time to some extent.

The TSP, with some minor modifications, can be used to model the microarray gene ordering (MGO) problem. In order to determine functional relationships between groups of genes that are often co-regulated and involved in the same cellular process, gene ordering is necessary. Gene ordering provides a sequence of genes such that those that are functionally related are closer to each other in the ordering [25]. This functional relationship among genes is determined by microarray gene expression levels. A microarray is typically a glass slide, onto which thousands of genes are attached at fixed locations (spots). By performing biological experiments gene expression levels are obtained from microarray [26]. A good solution of the gene ordering problem (i.e., finding optimal order of large DNA microarray gene expression data) has similar genes grouped together in clusters. Similarity between genes can be measured in terms of Euclidean distance, Pearson correlation, absolute correlation, Spearman rank correlation, etc. Investigations for clustering gene expression profiles include complete and average linkage (different versions of hierarchical clustering) [25, 27], self-organizing maps (SOM) [28] and Genetic Algorithms [29, 30].

Tsai et al. [29] formulated the MGO problem as TSP and applied family competition GA (FCGA) for solving it. They associated one imaginary city to each gene, and obtain the distance between any two cities (genes) from the matrix of inter gene distances. For microarray gene ordering it is necessary to minimize the distance between the genes that are in the neighborhood of each other, not the distant genes. However, Tsai et al. tried to minimize the distance between distant genes as well [29, 30]. This problem for TSP formulation in microarray gene ordering using GA is minimized in NNGA [30], where relatively long distances between genes are ignored for fitness evaluation. The present investigation has three parts. First, we define a new nearest fragment operator (NF) and a modified version of order crossover operator (viz., modified order crossover, MOC). The NF reduces the limitation of NN heuristic in path construction. This reduction is achieved by determining optimum number of fragments in terms of the number of cities and then greedily reconnecting them. The nearest fragment operator also takes care of the neighbor genes not the distant ones for MGO and provides good results without ignoring any long distances between genes for fitness evaluation. The modified version of order crossover operator (MOC) handles the indefinite computational time due to random length of substring and its random insertion in order crossover. This is done by systematically determining a appropriate substring length from the parent chromosome for performing crossover. While the position of the substring in the parent chromosome is chosen randomly, the length of the substring is predetermined in MOC. In the second part of the investigation, the effectiveness of the new operators for solving TSP is established. Finally, in the third part the microarray gene ordering problem is considered. Comparison of the proposed genetic operators is carried out with other techniques based on GAs, neural networks and clustering in terms of a biological score.

In Section 2 we provide, in brief, a formal definition of TSP and relevance of TSP in microarray gene ordering. The different components of GAs along with their implementation

for solving TSP are discussed in Section 3. New operators such as NF and MOC, and the algorithm based on them for TSP and gene ordering are described in Section 4. Then we present in Section 5 the results obtained with our algorithm for different TSP instances. Section 6 concludes the investigation.

2 TSP definition and relevance in microarray gene ordering

Let $\{1, 2, \ldots, n\}$ be the labels of the n cities and $C = [c_{i,j}]$ be an $n \times n$ cost matrix where $c_{i,j}$ denotes the cost of traveling from city i to city j. The Traveling Salesman Problem (TSP) is the problem of finding the shortest closed route among n cities, having as input the complete distance matrix among all cities. The total cost A of a TSP tour is given by

$$A(n) = \sum_{i=1}^{n-1} C_{i,i+1} + C_{n,1} \tag{1}$$

The objective is to find a permutation of the n cities, which has minimum cost.

An optimal gene order, a minimum sum of distances between pairs of adjacent genes in a linear ordering $1, 2, \ldots, n$, can be formulated as [25]

$$F(n) = \sum_{i=1}^{n-1} C_{i,i+1}, \tag{2}$$

where n is the number of genes and $C_{i,i+1}$ is the distance between two genes i and $i + 1$. In this study, the Euclidean distance is used to specify the distance $C_{i,i+1}$.

Let $X = x_1, x_2, \ldots, x_k$ and $Y = y_1, y_2, \ldots, y_k$ be the expression levels of the two genes in terms of log-transformed microarray gene expression data obtained over a series of k experiments. The Euclidean distance between X and Y is

$$C_{x,y} = \sqrt{\{x_1 - y_1\}^2 + \{x_2 - y_2\}^2 + \cdots + \{x_k - y_k\}^2}. \tag{3}$$

One can thus construct a matrix of inter-gene distances, which serves as a knowledge-base for mining gene order using GA. Using this matrix one can calculate the total distance between adjacent genes and find that permutation of genes for which the total distance is minimized. This is analogous to the traveling salesman problem. One can simply associate one imaginary city to each gene, and obtain the distance between any two cities (genes) from the matrix of inter gene distances. The formula (Eq. (2)) for optimal gene ordering is the same as used in TSP, except the distance from the last gene to first gene, which is omitted, as the tour is not a circular one.

```
begin GA
    Create initial population
    while generation_count < k do
        /* k = max. number of generations. */
        begin
            Selection and Elitism
            Produce children by crossover from
                                -selected parents
            Mutate the individuals
            Increment generation_count
        end
        Output the best individual found
end GA
```

Fig. 1 The Pseudo-code of Genetic Algorithm (GA)

3 Genetic algorithms for solving TSP and MGO

Genetic algorithms (GAs) [3] are randomized search and optimization techniques guided by the principles of evolution and natural genetics, and have a large amount of implicit parallelism. GAs perform multimodal search in complex landscapes and provide near optimal solutions for objective or fitness function of an optimization problem. In GAs, the parameters of the search space are encoded in the form of strings (chromosomes). A collection of such strings is called a population. Initially a random population is created, which represents different points in the search space. Based on the principle of survival of the fittest, a few among them are selected and each is assigned a number of copies that go into the mating pool. Biologically inspired operators like mating, crossover, and mutation are applied on these strings to yield a new generation of strings. The process of selection, crossover and mutation continues for a fixed number of generations or until a termination condition is satisfied. A general description of Genetic Algorithm is presented in this section for solving TSP using elitist model. Roughly, a genetic algorithm works as follows (see Fig. 1):

3.1 Chromosome representation and nearest-neighbor heuristic

Various representation methods are used to solve the TSP problem using GA. Some of these are binary representation, path representation, matrix representation, adjacency representation, ordinal representation [1]. In order to find the shortest tour for a given set of n cities using GAs, the path representation is more natural for TSP [1]. We have used this representation in our proposed GA. In path representation, the chromosome (or, string) corresponding to a TSP tour is an array of n integers which is a permutation of $(1, 2, \ldots, n)$, where an entry i in position j indicates that city i is visited in the jth time instant. The objective is to find a string with minimum cost.

For solving TSP, the nearest neighbor tour construction heuristic is a common choice to construct the initial population. The salesman starts at some random city and then visits the city nearest to the starting city. From there he visits the nearest city that was not visited so far, until all the cities are visited, and the salesman returns to the starting city. The NN tours have the advantage that they only contain a few severe mistakes, while there are long segments connecting nodes with short edges. Therefore such tours can serve as good starting tours for subsequent refinement using other sophisticated search methods. In NN the main disadvantage is that, several cities are not considered during the course of the algorithm and have to be inserted at high costs in the end. This leads to severe mistakes in path construction. To overcome the disadvantages of the NN heuristics, we propose a new heuristic operator, called the Nearest Fragment (NF) operator (discussed in Section 4). However, unlike NN heuristic that is used only for constructing the initial population, NF is used in every generation (iteration) of GA with a predefined probability for every chromosome in the population as a subsequent tour improvement method.

3.2 Selection and elitism

A number of different selection implementations have been proposed in the literature [3], such as roulette wheel selection, tournament selection, linear normalization selection. Here linear normalization selection, which has a high selection pressure [3], has been implemented. In linear normalization selection, an individual is ranked according to its fitness, and then it is allowed to generate a number of offspring proportional to its rank position. Using the rank position rather than the actual fitness values avoids problems that occur when fitness values are very close to each other (in which case no individual would be favored) or when an extremely fit individual is present in the population (in such a case it would generate most of the offspring in the next generation). This selection technique pushes the population toward the solution in a reasonably fast manner, avoiding the risk of a single individual dominating the population in the space of one or two generations.

A new population is created at each generation (iteration) and after selection procedure, chromosome with highest fitness (least cost) from the previous generation replaces randomly a chromosome from this new generation provided fitness of the fittest chromosome in the previous generation is higher than the best fitness in this current generation in the elitist model.

3.3 Crossover

As the TSP is a permutation problem, it is natural to encode a tour by enumerating the city indices in order. This approach has been dominant in GAs for solving the TSP. In such an encoding, the chromosomal location of a city is not fixed, and only the sequence is meaningful. Some representative crossovers performed on order-based encodings include cycle crossover [21], partially matched crossover [3] and order crossover [3, 20]. Order crossover has been found to be one of the best in terms of quality and speed [1], and yet is simple to implement. Below order crossover is described briefly.

Order crossover (OC). The order crossover operator [3, 20] selects at random a substring in one of the parent tours, and the order of the cities in the selected positions of this parent is imposed on the other parent to produce one child. The other child is generated in an analogous manner for the other parent. As an example consider two parents A and B, and a substring in A of length 3, selected randomly, as shown [3].

$A = 1\ 2\ 3\ |5\ 6\ 7|\ 4\ 8\ 9\ 0$

and

$B = 8\ 7\ 1\ |2\ 3\ 0|\ 9\ 5\ 4\ 6$

The cities in the selected substring in A (here, 5, 6, and 7) are first replaced by $*$ in the receptor B.

$A = 1\ 2\ 3\ |5\ 6\ 7|\ 4\ 8\ 9\ 0$

and

$B = 8\ *\ 1\ |2\ 3\ 0|\ 9\ *\ 4\ *$

Now to preserve the relative order in the receiver, a sliding motion is made to leave the holes in the matching section marked in the receiver. The convention followed in [3] is to start this sliding motion in the second crossover site, so after the rearrangement we have

$A = 1\ 2\ 3\ |5\ 6\ 7|\ 4\ 8\ 9\ 0$

and

$B = 2\ 3\ 0\ |*\ \ *\ \ *|\ 9\ 4\ 8\ 1$

After that, the stars are replaced with the city names taken from the donor A resulting in the offspring B1

$B1 = 2\ 3\ 0\ |5\ 6\ 7|\ 9\ 4\ 8\ 1$

Similarly the complementary crossover from B to A yields

$A1 = 5\ 6\ 7\ |2\ 3\ 0|\ 4\ 8\ 9\ 1$

In order crossover (OC) the length of the substring for crossover (chosen from the parent string) is random and may often be significantly large; this can have an adverse impact on the computational time. This uncertainity is tackled with a small and predefined length of substring, obtained after extensive empirical studies, for crossover (discussed in Section 3).

3.4 Mutation

For TSP, the simple inversion mutation (SIM) is one of the leading performers [1]. Here simple inversion mutation (SIM) is performed on each string probabilistically as follows: Select randomly two cut points in the string, and reverse the substring between these two cut points. For example consider the tour

(1 2 |3 4 5| 6 7 8)

and suppose that the first cut point is chosen randomly between 2nd city and 3rd city, and the second cut point between the 5th city and the 6th city as shown. Then the resulting string will be

$C = (1\ 2\ |5\ 4\ 3|\ 6\ 7\ 8)$

4 New operators of GA

In this section, some new operators of GAs for solving TSP and microarray gene ordering are described. These are nearest fragment (NF) and modified order crossover (MOC). The genetic algorithm designed using these operators is referred to as FRAG_GA. The structure of the proposed FRAG_GA is provided in Fig. 2.

```
begin FRAG_GA
    Create initial population with Nearest-Neighbor Heuristic
    while generation_count < k do
        /* k = max. number of generations. */
        begin
            Apply NF heuristic or (NF and LK) heuristic
            Elitism
            Linear Normalized Selection
            MOC
            Mutation
            Increment generation_count
        end
        Output the best individual found
end FRAG_GA
```

Fig. 2 The Pseudo-code for FRAG_GA

The basic steps of the FRAG_GA are as follows:

Step 1. Create the string representation (chromosome of GA) for a TSP tour (an array of n integers), which is a permutation of $1, 2, \ldots, n$ with Nearest-Neighbor heuristic. Repeat this step to form the population of GA.

Step 2. The NF heuristic is applied on each chromosome probabilistically.

Step 3. Each chromosome is upgraded to local optimal solution using chained LK heuristic probabilistically. (If Step 3. is used in the GA we denote it as FRAG_GALK and otherwise as FRAG_GA.).

Step 4. Fitness of the entire population is evaluated and elitism is used, so that the fittest string among the child population and the parent population is passed into the child population.

Step 5. Using the evaluated fitness of entire population, linear normalized selection procedure is used.

Step 6. Chromosomes are now distributed randomly. Modified Order Crossover operator is applied between two consecutive chromosomes probabilistically.

Step 7. Simple inversion mutation (SIM) is performed on each string probabilistically.

Step 8. Generation count of GA is incremented and if it is less than the maximum number of generations (predefined) then from Step 2 to Step 6 are repeated.

Local search heuristics, such as 2-swap, 2-opt [19], 3-opt [19], and Lin-Kernighan (LK) heuristic [12–14, 16], have been extensively applied in GAs for solving TSPs. These techniques exchange some edges of parents to generate new children. Usually, stronger local search methods correspond to better performing GAs. The mechanisms by which these methods add and preserve edges vary. 2-swap arbitrarily changes two cities at a time, removing four edges at random and adding four edges at random. 2-opt, 3-opt and LK exchange edges if the generated solution is better than the original one. In each iteration, 2-opt and 3-opt exchange two and three edges respectively, while, LK exchanges a variable number of edges. In the present investigation Concorde version of chained-LK [31] is used for fair comparison with [16]. In the following sections, the new operators NF and MOC are described in details.

4.1 Nearest fragment heuristic (NF)

In this process, each string (chromosome in GA) is randomly sliced in *frag* fragments. The value of *frag* is determined by FRAG_GA in terms of the total no. of cities/genes (n) for a particular TSP instance (or microarray data). The systematic process of determining *frag* is described later in this section. As an example, let us consider a string P that is sliced into three random fragments (1–8), (9–14) and (15–20) for a

20-city problem.

$P = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ |9\ 10\ 11\ 12\ 13\ 14|\ 15\ 16\ 17\ 18\ 19\ 20$

For tour construction the first fragment (9–14) is chosen randomly. From the last city of that fragment (14) the nearest city that is either a start or an end point of a not yet visited tour fragment is determined from the cost matrix. In this example, let the nearest city (among 1, 8, 15 and 20) be 20. The fragment containing the nearest city is connected to the selected fragment, with or without inversion depending on whether the nearest city is the last city of a fragment or not respectively. In this example , the fragment 15–20 is inverted and connected to fragment 9–14, resulting in the following partial tour $P1$.

$P1 = 9\ 10\ 11\ 12\ 13\ 14\ |20\ 19\ 18\ 17\ 16\ 15$

The process is repeated until all fragments have been reconnected. From the last city (15) of $P1$ the nearest city from unvisited fragment (1–8) is say 1. From this result the final string $P2$, shown below, is formed.

$P2 = 9\ 10\ 11\ 12\ 13\ 14\ 20\ 19\ 18\ 17\ 16\ 15\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$

The basic steps of choosing frag value systematically for a TSP instance with n cities are:

Step (1) Set *frag* value to $frag_{min}$.
Step (2) Run FRAG_GA with the selected *frag* value for x generations and store the number of times the best tour cost is decreased from one generation to the next for that *frag* value. Denote the stored values by $Decr_{cost}$.
Step (3) Increase *frag* by amount $\Delta frag$.
Step (3) Repeat Step 2 to 3 until $frag <= frag_{max}$
Step (4) Find θ consecutive *frag* values for which the summation of corresponding $Decr_{cost}$ values is maximum.
Step (5) The best *frag* value is set to the average of the selected five consecutive *frag* values.
Step (6) Repeat Step 1 to 5 ten times and the average of the best *frag* values is fixed as the final *frag* value for NF for a particular TSP instance.

In this study we have used $frag_{min} = \frac{n}{50}$, $frag_{max} = \frac{n}{2}$, $x = \frac{n}{10}$, $\Delta frag = \frac{n}{50}$, and $\theta = 5$, though experiments were conducted for a few other values as well with similar results. The value of $frag_{max}$ is not set to n as this will lead to NN heuristic. Also, the crossover operator was disabled. The motivation for setting the initial frag value to a low one (and consequently fragment lengths are larger) and then increasing it is that, first exploring the distant neighbors reduces the chances of locking at a local optimal tour for the GA. The probabilistic use of NF also helps to come out from local optimal solution by leaving some chromosomes for mutation and crossover operators to explore.

As an example, consider a 100 city problem with $frag = \frac{n}{16}$, and consequently the fragment length is 16 on an average. As the initial population of the FRAG_GA is formed with NN heuristic there is a likelihood that a city at one end of a fragment is close to the 16th neighbor of the similar end of the next/previous fragment and consequently, they may be connected by the NF heuristic. In the later generations of the GA, using $\frac{n}{16}$ fragments in NN heuristic, explores on an average, from any city to the 16th city in the chromosome rather than the 16th neighbor. Due to random slicing of the chromosome, some fragment lengths will be obviously greater than 16 and some less than 16, and consequently different types of neighbors will be considered. The lowest frag value is set to $\frac{n}{16}$ from the studies in [14], where it is mentioned that good/optimal results are obtained for most of the TSP instances in the TSP library [32] with a search space near about 16 neighbors. The more distant neighbors are mostly explored with mutation and crossover operators.

4.2 Modified order crossover (MOC)

As already mentioned, in order crossover the length of a substring is chosen randomly and can lead to an increase in the computational time, this uncertainty can be minimized if the length of the substring for performing crossover can be fixed to a small value. However, no study has been reported in the literature for determining an appropriate value of the length of a substring for performing order crossover. Such an attempt is made in this article for finding a small substring length for MOC that provides good results for TSP/microarray data with the lowest computational cost.

Unlike order crossover, where the substring length is randomly chosen, in MOC substring length is determined automatically by the FRAG_GA in a similar way of choosing frag value in Section 4.1. In the process of choosing appropriate substring length, NF heuristic is also present in FRAG_GA with its final frag value. As final frag value for NF is determined without any crossover operator, it is preferable to start the process of choosing substring length for MOC initially with a very small value like $\frac{n}{32}$ (very close to no MOC) and then increasing it. For example, for a 10 city problem let the systematically chosen substring length by FRAG_GA is 2. Now for the parents A and B the chromosomes may be as follows

$A = 0\ 9\ 8\ 4\ |5\ 6|\ 7\ 3\ 2\ 1$

and

$B = 9\ 5\ 4\ 1\ |2\ 3|\ 0\ 6\ 8\ 7$

The cities in the selected substring in A (here, 5 and 6) are first replaced by $*$ in the receptor B.

$B = 9\ *\ 4\ 1\ |2\ 3|\ 0\ *\ 8\ 7$

Now to preserve the relative order in the receiver, the convention followed in [3] is to gather the holes in the second crossover site and insert the substring there. But this convention leads to loss of information and increases randomness in the receiver because, after insertion of substring 56 in B neither 5 is nearer to 3, nor 6 is nearer to 0. To reduce this randomness, in MOC the holes are gathered in the position of the last deleted city (here city 6) of the receiver B.

$B = 9\ 4\ 1\ 2\ 3\ 0\ |\ *\ *|\ 8\ 7$

So after substring insertion, B is as follows:

$B = 9\ 4\ 1\ 2\ 3\ 0\ 5\ 6\ 8\ 7$

Now, at least one edge of the substring is nearer to the next city (city 6 is nearer to 8 according to chromosome B, and this information is preserved). Same convention is followed for inserting substring in chromosome A.

5 Experimental results

FRAG_GA is implemented in C on Pentium-4 (1.2 GHz) and the results are compared with those obtained using SWAP_GATSP [6], MMGA [4], IGA [5], OX_SIM, (standard GA with order crossover and simple inversion mutation) [1] MOC_SIM (Modified order crossover and SIM), and self organizing map (SOM) [10] for solving TSP. For fair comparison with the above mentioned methods Lin-Kernighan (LK) heuristic is not used with FRAG_GA, whereas, for comparison with HeSEA [16] and other LK based methods each chromosome in FRAG_GALK is updated probabilistically with 20 runs of consecutive chained LK and mutation (as recommended in [16]). Several benchmark TSP instances, for which the comparative study with various recently developed pure genetic algorithms (without LK), and SOM are available in the literature, are taken from the TSPLIB [32] without any bias on data sets. These include Grtschels24.tsp, bayg29.tsp, Grtschels48.tsp, eil51.tsp, St70.tsp, eil76.tsp, kroA100.tsp, d198.tsp, ts225.tsp, pcb442.tsp and rat783.tsp. For comparative study between HeSEA, FRAG_GALK, and other LK based methods the available TSP instances are lin318, rat783, pr1002, vm1084, pcb1173, u1432, u2152, pr2392, pcb3038, fnl4461, and usa13509. For biological microarray gene ordering, Cell Cycle cdc15, Cell Cycle and Yeast Complexes datasets are chosen [33]. The three data sets consists of about 782, 803 and 979 genes respectively, which are cell cycle regulated in Saccharomyces cerevisiae, with different number of experiments (24, 59 and 79 respectively) [26]. Each dataset is classified into five groups termed G1, S, S/G2, G2/M, and M/G1 by Spellman et al. [26]. Results are compared with those obtained using GAs [29, 30], different versions of hierarchical clustering [25, 27] and self-organizing map (SOM) [28] for solving microarray gene ordering. Throughout the experiments for FRAG_GA, SWAP_GATSP, OX_SIM, and MOC_SIM the population size is set equal to 100. Crossover probability is fixed at 0.6 and mutation probability is fixed at 0.02 across the generations. For FRAG_GA and FRAG_GALK the probability of applying NF heuristic is fixed at 0.3. Using these parameters FRAG_GA first systematically determines and stores the appropriate *frag* value for NF heuristic and substring length for MOC for each problem instance in a way mentioned in Sections 4.1 and 4.2, and then with these values, tour cost is optimized. Table 1 shows the various parameters of different genetic algorithms used in this current investigation.

Table 1 Different parameters of FRAG_GA, SWAP_GATSP, OX_SIM, and MOC_SIM

Population size	NF Probability for FRAG_GA	Crossover probability	Mutation probability
100	0.3	0.6	0.02

First, we provide results comparing our method (FRAG_GA) with other methods that do not use LK heuristics and then comparisons of results are provided with our method incorporating LK heuristic (FRAG_GALK) with other LK based methods .

5.1 Comparison with other GA approaches for TSP

Table 2 summarizes the results obtained over 30 runs by running the FRAG_GA, SWAP_GATSP [6], OX_SIM and MOC_SIM [1] on the aforesaid eleven different TSP instances. For SWAP_GATSP, OX_SIM and MOC_SIM, the overlapping parameters (Table 1) are taken from FRAG_GA. For each problem the total number of cities and the optimal tour cost are mentioned below the problem name in the first column. The total number of generations in which the best result and the average result are obtained is mentioned in columns 3–6 within parentheses. The error percentages are shown in third row for each problem, where the error percentage is defined as

$$E = \frac{average - optimal}{optimal} \times 100. \tag{4}$$

Experimental results (Tables 2 and 3) using FRAG_GA are found to be superior in terms of quality of solution (best result, average result and error percentage) with less number

Table 2 Comparison of the results over 30 runs obtained using FRAG_GA, SWAP_GATSP, OX_SIM, and MOC_SIM for different TSP instances

Problem		FRAG_GA	SWAP_GATSP	OX_SIM	MOC_SIM
Grtschels24	best	1272 (13)	1272 (50)	1272 (800)	1272 (600)
24	average	1272 (100)	1272 (200)	1322 (1500)	1272 (1500)
1272	error (%)	0.0000	0.0000	3.9308	0.0000
Bayg29	best	1610 (30)	1610 (60)	1620 (1000)	1610 (700)
29	average	1610 (100)	1615 (200)	1690 (1500)	1622 (1500)
1610	error (%)	0.0000	0.3106	4.9689	0.7453
Grtschels48	best	5046 (40)	5046 (200)	5097 (2500)	5057 (1700)
48	average	5054 (150)	5110 (700)	5410 (3000)	5184 (3000)
5046	error (%)	0.1585	1.2683	7.2136	2.7348
eil51	best	426 (45)	439 (220)	493 (2500)	444 (1600)
51	average	432 (150)	442 (700)	540 (3000)	453 (3000)
426	error (%)	1.4085	3.7559	26.7606	6.3380
St70	best	675 (40)	685 (600)	823 (4500)	698 (4500)
70	average	679 (150)	701 (1000)	920 (7500)	748 (7500)
675	error (%)	0.5926	3.8519	36.2963	10.8148
eil76	best	538 (75)	548 (700)	597 (5000)	562 (3800)
76	average	544 (150)	555 (1000)	620 (7500)	580 (7500)
538	error (%)	1.1152	3.1599	15.2416	7.8067
KroA100	best	21282 (80)	21397 (2000)	21746 (10000)	21514 (8200)
100	average	21303 (500)	21740 (3000)	22120 (12000)	21825 (12000)
21282	error (%)	0.0987	2.1521	3.9376	2.5515
d198	best	15780 (850)	15980 (4000)	16542 (10000)	16122 (9000)
198	average	15865 (2000)	16106 (4000)	17987 (16000)	16348 (16000)
15780	error (%)	0.5387	2.0659	13.9861	3.5995
ts225	best	126643 (1000)	127012 (4000)	135265 (10000)	128994 (10000)
225	average	126778 (2000)	128467 (4000)	138192 (16000)	130994 (16000)
126643	error (%)	0.1066	1.4403	9.1193	3.4356
pcb442	best	50778 (1900)	52160 (8000)	53320 (16000)	52852 (13000)
442	average	50950 (4000)	53800 (8000)	56330 (26000)	54173 (26000)
50778	error (%)	0.3387	5.9514	10.9339	6.6860
rat783	best	8850 (7500)	9732 (12000)	10810 (28000)	10155 (20000)
783	average	9030 (16000)	10087 (16000)	11136 (40000)	10528 (40000)
8806	error (%)	2.5437	14.5469	26.4592	19.5548

of generations when compared with those of other existing GAs [1, 4–6]. It is evident from the table that for different TSP instances the error percentages are lowest for FRAG_GA and the error percentages for MOC_SIM is much less than OX_SIM . The average of error percentages over all the TSP instances for MOC_SIM is 5.8425, which is also less than 14.4407 of OX_SIM. The error averages clearly indicates that the modification of order crossover improves its performance significantly over the existing order crossover which uses random substring length and its random insertion. The average of error percentages over all the TSP instances for FRAG_GA and SWAP_GATSP are 0.6274 and 3.5003 respectively.

Figure 3 shows a comparison of FRAG_GA, SWAP_GATSP and OX_SIM when the fitness value of the fittest string is plotted with iteration. The three programs were run for 12000 iterations for kroa100.tsp with population 100. At any iteration, the FRAG_GA has the lowest tour cost. It took 15.36 seconds, 19.94 seconds and 15.14 seconds by FRAG_GA, SWAP_GATSP and OX_SIM respectively for executing 12000 iterations. Moreover, only FRAG_GA is seen to converge at around 250 iterations at the optimal cost value of 21282 km. On the other hand, the cost is 21397 km for SWAP_GATSP after 3100 iterations and 21990 km for OX_SIM even after 12000 iterations.

Table 3 Average results for various GAs

Problem	Optimal	IGA	MMGA	SOM	FRAG_GA
eil51	426	499	446	432	432
st70	675	–	–	683	679
eil76	538	611	568	556	544
Kroa100	21282	24921	22154	–	21303
d198	15780	17925	16360	–	15865
ts225	126643	135467	129453	–	126778
pcb442	50778	59380	55660	55133	50950

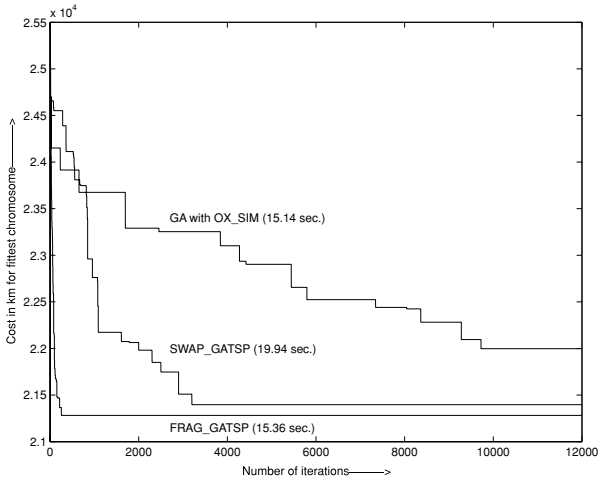

Fig. 3 Cost of fittest string Vs. Iteration for kroa100.tsp

Note that FRAG_GA takes almost the same time as OX_SIM using one more operator (NF), but provides better result in less number of paths. It is further to be pointed out that the NF operator creates an overhead, leading to an increase in the computation time for FRAG_GA, as compared to OX_SIM. However, this is compensated by the gain obtained in using the proposed MOC operator. As a consequence, the time required to execute one iteration, on an average, becomes almost equal for both FRAG_GA and OX_SIM. Similar observations are also made when the proposed method is compared with other GAs [1, 6] and other methods like Self Organizing Map (SOM) [10].

In Table 3 average results of FRAG_GA are compared to other GA based approaches viz., IGA and MMGA (whose results are taken from [4]) and Self Organizing Map (SOM) [10]. As can be seen from the table, the proposed approach is again found to consistently outperform IGA, MMGA, and SOM.

5.2 Comparison with other LK based approaches for TSP

Table 4 summarizes the results obtained over 20 runs by running the FRAG_GALK on different TSP instances mentioned in first column. 20 runs of LK [31] and mutation are applied on randomly chosen 50 chromosomes (among those who are not operated with NF heusristic) in each generation of FRAG_GALK. For HeSEA (with LK) [16], LKH (Multitrial LK) [13], iterated LK (ILK) [19], and tabu search with LK [8] the results are taken from [16]. While FRAG_GALK, HeSEA, LKH, and concorde chained LK (concorde) [31] are executed on Pentium-4 (1.2 GHz) personal computer, ILK and tabu search with LK are executed on Silicon Graphics 196 MHz MIPS R1000 and Pentium III 800 MHz respectively in [16]. For fair comparison Concorde chained LK is executed separately for same time as FRAG_GALK, but on average concorde converged to the mentioned solutions (in terms of error) before the allocated time. The total number of cities and the optimal tour cost are mentioned below the problem name in the first column. The error percentages (Equation 4) are shown in first row for each problem. The average number of generations over 20 runs for which the error percentages are obtained is mentioned in second row for each TSP instance. The third row for each TSP instance shows the average time in seconds taken by each method. From the table it is clear that FRAG_GALK produces comparable results with HeSEA with same version of LK in less computational time, whereas the quality of solution of FRAG_GALK is better than other algorithms with comparable computational time. So FRAG_GALK seems to be a better TSP solver among the existing ones. The time gain obtained by FRAG_GALK over HeSEA is due to probabilistic single run of computationally effective NF heuristic and MOC over each chromosome in FRAG_GALK, whereas, HeSEA uses 20 runs of edge-assembly crossover between the selected chromosomes and for all possible combinations of chromosomes with probability 1. Generations of LKH, ILK, and tabu with LK are not available.

5.3 Results for microarray gene ordering

FRAG_GA is applied for ordering the genes based on their expression levels obtained from microarray datasets. Performance of FRAG_GA for gene ordering is compared with other methods based on GAs, clustering and neural networks. GA based investigations include NNGA [30] and FCGA [29] (discussed in Section 1). Clustering methods can be broadly divided into hierarchical and nonhierarchical clustering approaches. Hierarchical clustering approaches [25, 27] group gene expressions into trees of clusters. They start with singleton sets and keep on merging the closest sets until all the genes form a single cluster. Complete-linkage and average-linkage belong to this category of clustering technique, differing only in the way the distance between clusters is defined. Nonhierarchical clustering approaches separate genes into groups according to the degree of similarity (as quantified by Euclidian distances, Pearson correlation) among genes. The relationships among the genes in a particular cluster generated by nonhierarchical clustering methods are lost. Self-organizing map (SOM) [28], a particular class of neural network, performs nonhierarchical clustering.

Table 5 summarizes the results in terms of the sum of gene expression distances (Eq. (2)), by executing the FRAG_GA, complete linkage, average linkage and SOM on the three different microarray datasets (results in terms of sum of gene expression distance and code for NNGA and FCGA are not available in the literature [29, 30]). For FRAG_GA and SOM, best and average results obtained over 30 runs are provided,

Table 4 Average results for various LK based algorithms

Problem		FRAG_GALK	HeSEA+LK	LKH	Concorde	ILK	Tabu+LK
lin318	error (%)	0.0000	0.0000	0.1085	0.0000	–	–
318	generation	2.8	3.2	–	–	–	–
42029	time (sec.)	1.4	2.3	1.4	1.4	–	–
rat783	error (%)	0.0000	0.0000	0.0000	0.1761	–	–
783	generation	8.0	8.4	–	–	–	–
8806	time (sec.)	5.9	39.1	2.2	5.9	–	–
pr1002	error (%)	0.0000	0.0000	0.0000	0.0215	0.1482	0.8794
1002	generation	22.2	12.0	–	–	–	–
259045	time (sec.)	34.6	91.0	7.5	34.6	298.0	1211.4
vm1084	error (%)	0.0000	0.0000	0.0068	0.0172	0.0217	0.3932
1084	generation	23.6	10.2	–	–	–	–
239297	time (sec.)	34.2	80.6	12.6	34.2	377.0	597.0
pcb1173	error (%)	0.0000	0.0000	0.0009	0.0070	0.0088	0.6996
1173	generation	22.5	11.5	–	–	–	–
56892	time (sec.)	38.7	84.5	11.8	39.0	159.0	840.0
u1432	error (%)	0.0000	0.0000	0.0000	0.0153	0.0994	0.4949
1432	generation	22.0	11.0	–	–	–	–
152970	time (sec.)	37.7	107.0	6.9	38.0	224.0	775.0
u2152	error (%)	0.0000	0.0000	0.0495	0.0242	0.1743	0.7517
2152	generation	31.5	17.5	–	–	–	–
64253	time (sec.)	48.3	211.0	135.0	49.0	563.0	1624.0
pr2392	error (%)	0.0000	0.0000	0.0000	0.0294	0.1495	0.6492
2392	generation	25.0	14.5	–	–	–	–
378032	time (sec.)	46.6	208.0	26.2	47.0	452.0	1373.0
pcb3038	error (%)	0.0000	0.0000	0.0068	0.1123	0.1213	0.8708
3038	generation	120.6	29.7	–	–	–	–
137694	time (sec.)	245.0	612.0	226.0	219.0	572.0	1149.0
fnl4461	error (%)	0.0014	0.0005	0.0027	0.0734	0.1358	0.9898
4461	generation	265.0	67.8	–	–	–	–
182566	time (sec.)	519.0	2349.0	528.0	519.0	889.0	1018.0
usa13509	error (%)	0.0061	0.0074	0.0065	0.1201	0.1638	0.8897
13509	generation	1102.5	223.0	–	–	–	–
19982859	time (sec.)	19203.0	34984.0	19573.0	19203.0	10694.0	5852.0

Table 5 Comparison of the results over 30 runs in terms of sum of gene expression distances for microarray data using various algorithms

	Cell cycle cdc15		Cell cycle		Yeast complexes	
Algorithms	Best	Average	Best	Average	Best	Average
FRAG_GA	1272 (1690)	1278 (4000)	2349 (2320)	2362 (4000)	3382 (3890)	3396 (6000)
Complete-linkage	1419	1419	2534	2534	3634	3634
Average-linkage	1433	1433	2559	2559	3681	3681
SOM	1874 (100000)	1905 (100000)	3018 (100000)	3094 (100000)	4376 (200000)	4449 (200000)

whereas, for complete and average linkage results remain same for all runs. The genetic parameters for FRAG_GA are the same as used before (see Table 1). For FRAG_GA and SOM the total number of generations/iterations, for which the best and average results are obtained are mentioned in columns 2–7 within parentheses. From the table it is clear that FRAG_GA produces superior gene ordering than related methods in terms of sum of the gene expression distances.

A biological score, that is different from the fitness function, is used to evaluate the final gene ordering. The biological

Table 6 Comparison of the best results over 30 runs in terms of $S(N)$ values for microarray data

Algorithms	Cell cycle cdc15	Cell cycle	Yeast complexes
FRAG_GA	540	635	384
NNGA	539	634	384
FCGA	521	627	–
Complete-linkage	498	598	340
Average-linkage	500	581	331
SOM	461	578	306

score is defined as [29]

$$S(n) = \sum_{i=1}^{n-1} s_{i,,i+1}$$

where

$$s_{i,,i+1} = 1, \quad \text{if gene } i \text{ and } i+1 \text{are in the same group}$$
$$= 0, \quad \text{if gene } i \text{ and } i+1 \text{ are not in the same group}$$

Using this, a gene ordering would have a higher score when more genes within the same group are aligned next to each other. So higher values of $S(n)$ indicate better gene ordering. For example consider the genes YML120C, YJR048W, YMR002W and YDR432W belonging to groups G2/M, S/G2, S/G2 and G2/M respectively. In the above-mentioned ordering they will return a biological score of $0 + 1 + 0 = 1$, whereas if they are ordered like YJR048W, YMR002W, YDR432W and YML120C then the score will be $1 + 0 + 1 = 2$. The scoring function is therefore seen to reflect well the order of genes in biological sense. Note that, although $S(n)$ provides a good quantitative index for gene ordering, using it as the fitness function in GA based ordering is not practical, since the information about gene categories is unknown for most of the genes in the real world .

Table 6 shows the best results over 30 runs of the above methods in terms of $S(n)$ value, where larger values are better ($S(n)$ values for NNGA are FCGA are taken from [30]). It is clear that FRAG_GA and NNGA [30] are comparable and they both dominate others. Note that FRAG_GA is a conventional GA, while NNGA (hybrid GA) is a one using LK heuristic [12]. The main reason for the good results obtained by FRAG_GA is that, biological solutions of microarray gene ordering lie in more than one sub optimal point (in terms of gene expression distance) rather than one optimal point and there exists different gene orders with same biological score.

6 Discussion and conclusions

A new "nearest fragment operator" (NF) and a modified version of order crossover operators (MOC) of GAs are described along with demonstrating their suitability for solving both TSP and microarray gene ordering (MGO) problem. A systematic method for determining appropriate number of fragments in NF and appropriate substring length in terms of the number of cities/genes in MOC are also provided. These newly designed genetic operators showed superior performance on both TSP and gene ordering problem. The said operators are capable of aligning more genes with the same group next to each other compared to other algorithms, thereby producing better gene ordering. Infact, FRAG_GA produces comparable and sometimes even superior results than NNGA, a GA which implements Lin-Kernighan local search, for solving MGO problem in terms of biological score.

The representation used in the present investigation is a direct one (integer i = city/gene i) and also used in all other state-of-the-art TSP solvers using genetic algorithm and LK heuristic based approaches. An indirect representation, like offset-based representation, in general takes more computational time in representation, whereas, there is no chance for improving the solution quality over optimal results for most of the TSP instances.

An advantage of FRAG_GALK is that the quality of the solution seems to be more stable than that obtained by LKH and concorde chained LK, when used to solve the benchmark TSP problems. An evolutionary algorithm for solving combinatorial optimization problems should comprise mechanisms for preserving good edges and inserting new edges into offspring, as well as mechanisms for maintaining the population diversity. In the proposed approach, nearest fragment heuristic, modified order crossover, and LinKernighan local search preserve good edges and add new edges. The proposed method can seamlessly integrate NF, MOC, and LK to improve the overall search.

The present investigation indicates that incorporation of the new operators in FRAG_GA and LK in FRAG_GALK yield better results as compared to other pure GAs, Self Organizing Map, and related LK based TSP solvers. With its superior results in reasonable computation time FRAG_GALK can be considered as one of the state-of-the-art TSP solver.

Acknowledgment This work is partially supported by the grant no. 22(0346)/02/EMR-II of the Council of Scientific and Industrial Research (CSIR), New Delhi.

References

1. Larranaga P, Kuijpers C, Murga R, Inza I, Dizdarevic S (1999) Genetic algorithms for the traveling salesman problem: a review of representations and operators. Artificial Intell Rev 13:129–170
2. Garey MR, Johnson, DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman and Co., San Francisco

3. Goldberg DE (1989) Genetic algorithm in search, optimization and machine learning, Machine Learning, Addison-Wesley, New York
4. Tsai CF, Tsai CW, Yang T (2002) A modified multiple-searching method to genetic algorithms for solving traveling salesman problem. In: IEEE int conf systems, Man and cybernetics, vol. 3, pp 6–9
5. Jiao L, Wang L (2000) A novel genetic algorithm based on immunity. IEEE Transactions on Systems, Man and Cybernetics, Part A 30(5):552–561
6. Ray SS, Bandyopadhyay S, Pal SK (2004) New operators of genetic algorithms for traveling salesman problem. Cambridge, UK, ICPR-04 2:497–500
7. Fiechter CN (1994) A parallel tabu search algorithm for large traveling salesman problems. Discrete Appl Math Combin Oper Res Comput Sci 51:243–267
8. Zachariasen M, Dam M (1995) Tabu search on the geometric traveling salesman problem. In: Proc. of int conf on metaheuristics, pp 571–587
9. Potvin JY (1993) The traveling salesman problem: a neural network perspective. ORSA J Comput 5:328–348
10. Bai Y, Zhang W, Jin Z (2006) An new self-organizing maps strategy for solving the traveling salesman problem. Chaos, Solitons & Fractals 28(4):1082–1089
11. Stutzle T, Dorigo M (1999) ACO algorithms for the traveling salesman problem, evolutionary algorithms in engineering and computer science. John Wiley and Sons
12. Lin S, Kernighan BW (1973) An effective heuristic for the traveling salesman problem. Oper Res 21(2):498–516
13. Helsgaun K (2000) An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur J Oper Res 1:106–130
14. Applegate D, Cook W, Rohe A (2000) Chained Lin-Kernighan for large traveling salesman problems. Tech Rep Dept Comput Appl Math Rice Univ
15. Gamboa D, Rego C, Glover F (2006) Implementation analysis of efficient heuristic algorithms for the traveling salesman problem. Computers & Operations Res 33(4):1154–1172
16. Tsai HK, Yang JM, Tsai YF, Kao CY (2004) An evolutionary algorithm for large traveling salesman problems. IEEE Transactions on Systems, Man and Cybernetics, Part B: Cyebernetics 34(4):1718–1729
17. Reinelt G (1994) The traveling salesman: computational solutions for TSP applications. Lecture notes in computer science, Springer-Verlag 840
18. Bentley JL (1992) Fast algorithms for geometric traveling salesman problems. ORSA J Computing 4(4):387–411
19. Johnson DS, McGeoch LA (1996) The traveling salesman problem: a case study in local optimization. Local search in combinatorial optimization. Wiley and Sons, New York
20. Davis L (1985) Applying adapting algorithms to epistatic domains. In: Proc. int. joint conf. artificial intelligence, Quebec, canada
21. Oliver I, Smith D, Holland J (1987) A study of permutation crossover operators on the traveling salesman problem. Second int. conf. genetic algorithms, pp 224–230
22. Starkweather T, McDaniel S, Mathias K, Whitley D, Whitley C (1991) A comparison of genetic sequencing operators. 4th Int. conf. genetic algorithms, pp 69–76
23. Whitley D, Starkweather T, Fuquay D (1989) Scheduling problems and traveling salesman: the genetic edge recombination operator. 3rd Int. conf. genetic algorithms, pp. 133–140
24. Homaifar A, Guan S, Liepins G (1993) A new approach on the traveling salesman problem by genetic algorithms. 5th Int conf genetic algorithms, pp 460–466
25. Biedl T, Brejov B, Demaine ED, Hamel AM, Vinar T (2001) Optimal arrangement of leaves in the tree representing hierarchical clustering of gene expression data. Tech Rep 2001–14, Dept Computer Sci., Univ. Waterloo
26. Spellman PT, Sherlock G, Zhang MQ, Iyer VR, Anders K, Eisen MB, Brown PO, Botstein D, Futcher B (1998) Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisia by microarray hybridization. Molecular Biology Cell 9:3273–3297
27. Eisen MB, Spellman PT, Brown PO, Botstein D (1998) Cluster analysis and display of genome-wide expression patterns. In: Proc. national academy of sciences, vol. 95, pp 14863–14867
28. Tamayo P, Slonim D, Mesirov J, Zhu Q, Kitareewan S, Dmitrovsky E, Lander ES, Golub TR (1999) Interpreting patterns of gene expression with self-organizing maps: methods and application to hematopoietic differentiation. In: Proc. national academy of sciences, pp 2907–2912
29. Tsai HK, Yang JM, Kao CY (2002) Applying genetic algorithms to finding the optimal gene order in displaying the microarray data. GECCO, pp. 610–617
30. Lee SK, Kim YH, Moon BR, (2003) Finding the Optimal Gene Order in Displaying Microarray Data. GECCO, pp. 2215–2226
31. Applegate D, Bixby R, Chvtal V, Cook W (2003) Concorde package. [online]. www.tsp.gatech.edu/concorde/downloads/codes/src/co031219.tgz.
32. TSPLIB, http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/.
33. Website, http://www.psrg.lcs.mit.edu/clustering/ismb01/optimal.html.

Shubhra Sankar Ray is a Visiting Research Fellow at the Center for Soft Computing Research: A National Facility, Indian Statistical Institute, Kolkata, India. He received the M.Sc. in Electronic Science and M.Tech in Radiophysics & Electronics from University of Calcutta, Kolkata, India, in 2000 and 2002, respectively. Till March 2006, he had been a Senior Research Fellow of the Council of Scientific and Industrial Research (CSIR), New Delhi, India, working at Machine Intelligence Unit, Indian Statistical Institute, India. His research interests include bioinformatics, evolutionary computation, neural networks, and data mining.

Sanghamitra Bandyopadhyay is an Associate Professor at Indian Statistical Institute, Calcutta, India. She did her Bachelors in Physics and Computer Science in 1988 and 1992 respectively. Subsequently, she did her Masters in Computer Science from Indian Institute of Technology (IIT), Kharagpur in 1994 and Ph.D in Computer Science from Indian Statistical Institute, Calcutta in 1998.

She has worked in Los Alamos National Laboratory, Los Alamos, USA, in 1997, as a graduate research assistant, in the University of New South Wales, Sydney, Australia, in 1999, as a post doctoral fellow,

in the Department of Computer Science and Engineering, University of Texas at Arlington, USA, in 2001 as a faculty and researcher, and in the Department of Computer Science and Engineering, University of Maryland Baltimore County, USA, in 2004 as a visiting research faculty.

Dr. Bandyopadhyay is the first recipient of Dr. Shanker Dayal Sharma Gold Medal and Institute Silver Medal for being adjudged the best all round post graduate performer in IIT, Kharagpur in 1994. She has received the Indian National Science Academy (INSA) and the Indian Science Congress Association (ISCA) Young Scientist Awards in 2000, as well as the Indian National Academy of Engineering (INAE) Young Engineers' Award in 2002. She has published over ninety articles in international journals, conference and workshop proceedings, edited books and journal special issues and served as the Program Co-Chair of the 1st International Conference on Pattern Recognition and Machine Intelligence, 2005, Kolkata, India, and as the Tutorial Co-Chair, World Congress on Lateral Computing, 2004, Bangalore, India. She is on the editorial board of the International Journal on Computational Intelligence. Her research interests include Evolutionary and Soft Computation, Pattern Recognition, Data Mining, Bioinformatics, Parallel & Distributed Systems and VLSI.

Sankar K. Pal (www.isical.ac.in/∼sankar) is the Director and Distinguished Scientist of the Indian Statistical Institute. He has founded the Machine Intelligence Unit, and the Center for Soft Computing Research: A National Facility in the Institute in Calcutta. He received a Ph.D. in Radio Physics and Electronics from the University of Calcutta in 1979, and another Ph.D. in Electrical Engineering along with DIC from Imperial College, University of London in 1982.

He worked at the University of California, Berkeley and the University of Maryland, College Park in 1986-87; the NASA Johnson Space Center, Houston, Texas in 1990-92 & 1994; and in US Naval Research Laboratory, Washington DC in 2004. Since 1997 he has been serving as a Distinguished Visitor of IEEE Computer Society (USA) for the Asia-Pacific Region, and held several visiting positions in Hong Kong and Australian universities. Prof. Pal is a Fellow of the IEEE, USA, Third World Academy of Sciences, Italy, International Association for Pattern recognition, USA, and all the four National Academies for Science/Engineering in India. He is a co-author of thirteen books and about three hundred research publications in the areas of Pattern Recognition and Machine Learning, Image Processing, Data Mining and Web Intelligence, Soft Computing, Neural Nets, Genetic Algorithms, Fuzzy Sets, Rough Sets, and Bioinformatics.

He has received the 1990 S.S. Bhatnagar Prize (which is the most coveted award for a scientist in India), and many prestigious awards in India and abroad including the 1999 G.D. Birla Award, 1998 Om Bhasin Award, 1993 Jawaharlal Nehru Fellowship, 2000 Khwarizmi International Award from the Islamic Republic of Iran, 2000–2001 FICCI Award, 1993 Vikram Sarabhai Research Award, 1993 NASA Tech Brief Award (USA), 1994 IEEE Trans. Neural Networks Outstanding Paper Award (USA), 1995 NASA Patent Application Award (USA), 1997 IETE-R.L. Wadhwa Gold Medal, the 2001 INSA-S.H. Zaheer Medal, and 2005-06 P.C. Mahalanobis Birth Centenary Award (Gold Medal) for Lifetime Achievement.

Prof. Pal is an Associate Editor of IEEE Trans. Pattern Analysis and Machine Intelligence, IEEE Trans. Neural Networks [1994–98, 2003–06], Pattern Recognition Letters, Neurocomputing (1995–2005), Applied Intelligence, Information Sciences, Fuzzy Sets and Systems, Fundamenta Informaticae, Int. J. Computational Intelligence and Applications, and Proc. INSA-A; a Member, Executive Advisory Editorial Board, IEEE Trans. Fuzzy Systems, Int. Journal on Image and Graphics, and Int. Journal of Approximate Reasoning; and a Guest Editor of IEEE Computer.