Soft Comput (2006) 10: 1075–1090
DOI 10.1007/s00500-005-0044-4

ORIGINAL PAPER

Kit Yan Chan · M. Emin Aydin · Terence C. Fogarty

Main effect fine-tuning of the mutation operator and the neighbourhood function for uncapacitated facility location problems

Published online: 3 March 2006

Abstract In both genetic algorithms (GAs) and simulated annealing (SA), solutions can be represented by gene representation. Mutation operator in GA and neighborhood function in SA are used to explore the solution space. They usually select genes for performing mutation. The rate of selection of genes can be called mutation rate. However, randomly selecting genes may not be the best way for both algorithms. This paper describes how to estimate the main effect in genes representation. The resulting estimates cannot only be used to understand the domination of gene representation, but also employed to fine-tune the mutation rate in both the mutation operator in the GA and the neighborhood function in the SA. It has been demonstrated the use of the proposed methods for solving uncapacitated facility location problems and discuss the examination of the proposed methods with some useful comparisons with both the latest developed GA and SA for solving this problem. For many well-known benchmark problems, the proposed methods yield better results in solution quality than the previously used methods.

Keywords Mutation rate · Uncapacitated facility location problem · Genetic algorithm · Simulated annealing · Main effect

1 Introduction

Simulated annealings (SAs) [11] and genetic algorithms (GAs) [16] have become increasingly popular for searching good solutions in many hard combinatorial optimization problems. It can be found from many literatures [2,4,7,8,12,13,17,21] that they can find good or reasonable solutions, but in many cases they cannot reach the global optimum. Based on the observation that the improvement of these searching algorithms decreases gradually along the searching stages. The search improves in the earlier stages of the searching process, but it saturates or even terminates in the later stages. They behave like the traditional local searching methods that drop into a local optima and cannot escape from it. It is hard to obtain any large improvements by examining neighbouring solutions in the later stages of the search. Vaessens et al. [20] and Reeves [15] put these searching methods into the context of local search or neighbourhood search. In this paper we intend to apply the idea of fine-tuning in both SAs and GAs to improve the search of both algorithms in the later stage.

In SAs, the approaches of dynamic neighbourhood size [19,22,23] have been used as fine-tuning for exploiting the promising regions and trying to find a good near optimal configuration by decreasing neighbouring size in the later searching stages. In GAs, the idea of fine-tuning has been applied to the mutation operator with a dynamic mutation rate by Fogarty [10]. He demonstrated that increasing mutation rate across gene representation can improve the performance of the GA. His approach is that the GA has a higher chance to be fine-tuned for a better solution in the later stages of the search, when more mutation is performed on low significant genes to the fitness function.

An experimental design measure, main effect [14], has been proposed as a measure of significance of genes by Chan et al. [5]. Similar to Fogarty [10] developed dynamic mutation rate, the information of main effects of genes has been used to fine-tune the search in the GA [5,6]. If the main effect of gene is low, then the gene has low significance to the fitness function. By mutating more frequently on genes with low main effect, the GA is more likely to be fine-tuned to a better solution in the later stage of the search.

This paper extends our work [5,6] by integrating the idea of fine-tuning with the mutation operator in the GA and the neighbourhood function in the SA. A new fine-tuning technique is proposed by using the information of the main effects in genes. For the mutation operator in the GA, the mutation rates of genes across gene representation vary according to

K.Y. Chan · M.E. Aydin · T.C. Fogarty (✉)
London South Bank University,
Faculty of Business,
Computing and Information Management,
103 Borough Road, London,
SE1 0AA, UK
E-mail: fogarttc@lsbu.ac.uk

the main effects of genes. In the later generations, more mutations are performed on genes with low main effects since the search has a better chance to be fine-tuned to a better solution. Similar to the GA, the solution of the SA can also be represented in gene representation. Also the neighbourhood function explores the solution space by mutating genes. Therefore similar idea can also been applied on the neighourhood function in the SA. In the neighourhood function, genes across gene representation are selected randomly to mutate for exploring solution space. The selection of genes in SA is called mutation rate. In this approach, the mutation rates of genes across gene representation vary according to the main effects of genes. Genes with low main effects are mutated more frequently for fine-tuning to a better solution in the later stage of the search. At the same time, genes with high main effects are mutated less frequently than the ones with low main effects.

The proposed methods are then used to solve the uncapacitated facility location (UFL) problem [9]. This is the problem of determining the optimal number of facility echelons, the number and the location of the facilities in each facility echelon. This is an NP-hard problem, and therefore the larger the size of the problem, the harder to find the optimal solution and the longer to reach a reasonable result. The proposed approach is integrated into both the existing GA [12] and SA [12,24] for solving the UFL problem. By solving a set of benchmark problems [3,12], it is shown that the proposed method achieves better solution quality than the existing algorithms.

The rest of the paper is organised as follows. Section 2 defines and describes how to evaluate the main effect of a gene. Section 3 discusses the formulation of the UFL problem and how to evaluate the main effect on each facility of the problem. Sections 4 and 5 discuss how to use the main effect to modify the mutation operator in the GA and the neighbourhood function in the SA. Section 6 shows and discusses the experimental results of the proposed methods and the existing one. Finally a conclusion is given in Sect. 7.

2 Estimation of the main effect

Experimental design involves the study of the effects of variables in a model. Usually, we are interested in examining the main effect in each variable. The main effect is defined as the change in the response of a model produced by a change in a level of a variable. It is the effect of a particular variable alone averaged across the levels of other variables and it is an outcome that is a consistent difference between the levels of a variable. With main effect measures, the domination of each variable can be indicated [14].

Let $f(x_1, x_2, \ldots, x_n)$ be a function with n variables, where $x_i \in [D^i_{\min} \cdot D^i_{\max}]$ with $i = 1, 2, \ldots, n$. In experimental design, $D^i_{\min}$ and D^imax can be defined as the lower and the higher level of the ith variable respectively. Let $\sum_i$ be the full factorial set that contains all combinations of levels in variables, and 0 is set in the ith variable in all combinations, i.e.:

$$\sum_i = \left\{\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_n \,\middle|\, \alpha_{j\neq i} = D^i_{\min} \text{or} D^i_{\max}, \text{but} \alpha_i = 0\right\}, \quad (1)$$

where $i = 1, 2, \ldots, n$. Let δ_i be an element in Σ_i, i.e.: $\delta_i = \{\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_n\} \in \Sigma_i$ with $i = 1, 2, \ldots, n$. Let two vectors Δ_i and $\bar{\Delta}_i$ be: Δ_i, $\bar{\Delta}_i = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ with $\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n = 0$, but $\alpha_i = D^i_{\min}$ in Δ_i and $\alpha_i = D^i_{\max}$ in $\bar{\Delta}_i$, where $i = 1, 2, \ldots, n$.

The fitness difference of the ith variable is defined as:

$$d_i(\delta_i) = \left| f(\delta_i + \Delta_i) - f\left(\delta_i + \bar{\Delta}_i\right) \right|. \quad (2)$$

In Eq. (1), the ith variable α_i in δ_i is 0. The rest of the variables α_j in δ_i are randomly taken with either $D^i_{\min}$ or $D^i_{\max}$, where $j = 1, 2, \ldots, i-1, i+1, \ldots n$ but $j \neq i$.

The main effect of the ith variable of the function $f(x_1, x_2, \ldots, x_n)$ is defined to be:

$$M_i = \sum_{\delta_i \in \Sigma_i} d_i(\delta_i). \quad (3)$$

When the search space is too big for such a computation (as it is usually the case), the main effect M_i in Eq. (3) is approximated on a random sample.

For more details, the reader is referred to [14]. The main effect enables the gaining of further insights into the contribution of each variable in the function. It can reflect the contribution of the ith variable to the function. If the main effect is large, then the contribution of the ith variable to the function is large. In another extreme, if it is low, then its contribution to the function is low.

The main effect enables the gaining of further insights into the contribution of each variable in the function. As an example, three parametrical functions (F_1, F_2 and F_3) are discussed. Their variables have unequal contributions to the functions:

Function F_1:

Like in the sphere function, the variables increase their contribution to the function exponentially:

$$F_1(x) = \sum_{i=1}^{N} 1.5^{i-1} x_i^2,$$

where $x_i \in [-100, 100]$. The global minimum is 0.

Function F_2:

The Schwefel's function keeps pair-wise interactions among the variables:

$$F_2(x) = \sum_{i=1}^{N} \left(\sum_{j=1}^{i} x_j \right)^2$$

where $x_i \in [-100, 100]$. The global minimum is 0.

Function F_3:

The contribution of each variable is randomly distributed to the function:

$$F_3(x) = \sum_{i=1}^{N} r_i x_i^2$$

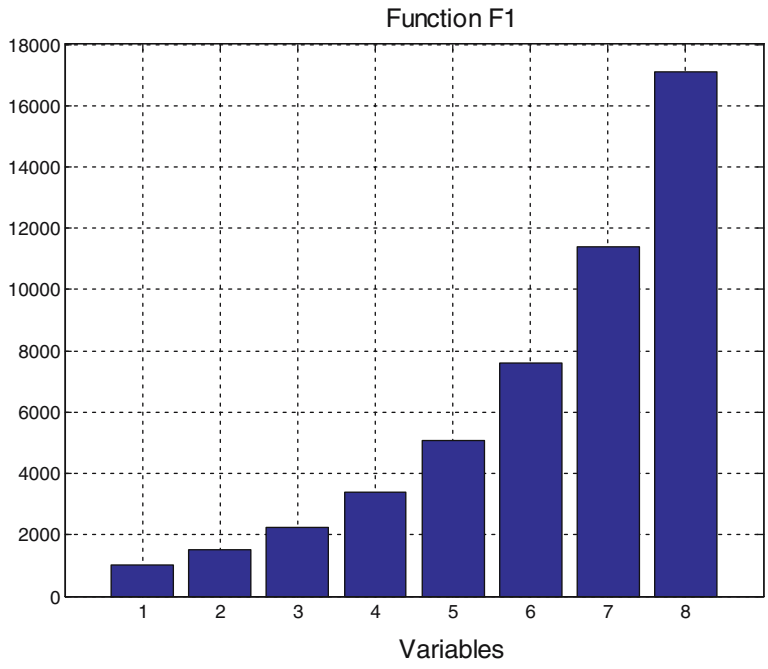

Fig. 1 Main effects of variables for function F_1

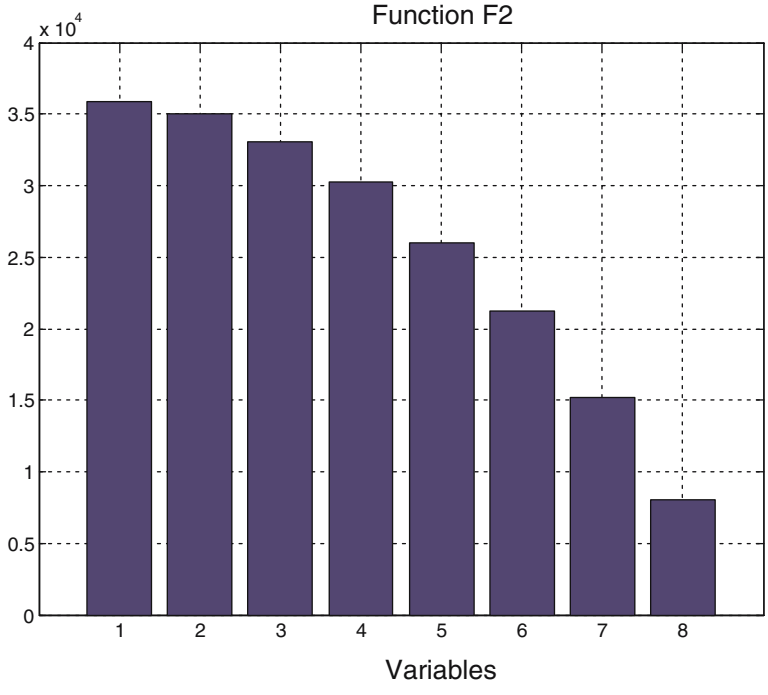

Fig. 2 Main effects of variables for function F_2

where r_i is a randomly shuffled array of integers between 1 to N and $x_i \in [-100, 100]$. The global minimum is 0.

The dimension of the functions is set to be $N = 8$. Equation (3) is used to estimate the main effect of each variable of the functions. The number of function evaluations used to estimate the main effect of each variable is set to 40. Therefore the total number of function evaluations used to estimate the main effects for all variables is 320 (i.e.: 40×8).

In these three functions, the magnitudes of main effects of variables are shown in Figs. 1, 2 and 3, thus the contribution of each variable to the function can be indicated. The x-axis shows the variable numbers and the y-axis shows the

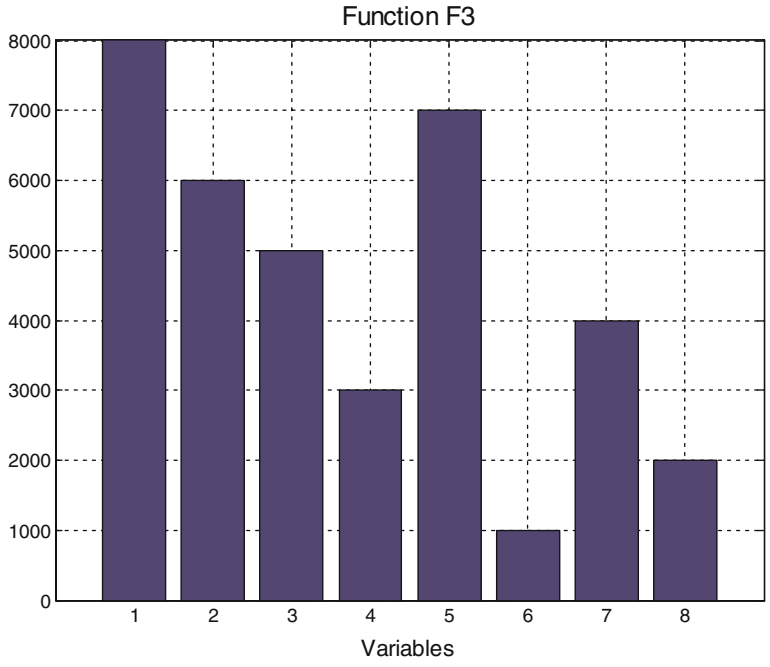

Fig. 3 Main effects of variables for function F_3

magnitude of main effect of each variable. It can be seen from Fig. 1 that the main effects of variables in function F_1 distribute in an ascending order. Figure 2 shows that the main effects of variables in function F_2 distribute in a descending order. Figure 3 shows that the main effects of variables in function F_3 distribute with no order. Therefore, these figures show that each variable has different contribution to the function.

3 Estimation of the main effect in UFL problems

In this section, the estimation of the main effect is illustrated in uncapacitated facility location UFL problems. UFL problems that have NP-Hard nature have been playing an important role in the operation research and the manufacturing context [9]. In a general form, the problem is to determine the optimal number of facility echelons, the number and location of facilities in each facility echelon, the assignment of distribution activities/commodities to the facilities, and the allocation of customers' demands among the facilities. UFL problems have several applications: the bank account location problem, the clustering problem, economic lot sizing, vehicle routing, network design, distributed data and communication networks.

The mathematical formulation of these problems as mixed integer programming models has proven very fruitfully in the derivation of solution methods. To formulate the UFL problems, consider a set of candidate sites for facility location, $J = \{1, \ldots, m\}$, and a set of customers, $I = \{1, \ldots, N\}$. Each facility $j \in J$ has a fixed cost f_j. Every customer $i \in I$ has a demand b_i, and c_{ij} is the unit transportation cost from facility j to customer i. Without a loss of generality one can normalise the customer demands to be $b_i = 1$. The problem is formulated in the following way:

$$Z = \min \left(\sum_{i=1}^{N} \sum_{j=1}^{m} c_{ij} x_{ij} + \sum_{j=1}^{m} f_j y_j \right) \tag{4}$$

subject to:

$$\sum_{j=1}^{m} x_{ij} = 1, \quad \text{for } \forall i \in I; \tag{5}$$

$$0 \leq x_{ij} \leq y_j \text{ and } y_j \in \{0; 1\}; \quad \text{for } \forall i \in I \text{ or } \forall j \in J; \tag{6}$$

where x_{ij} represents the quantity supplied from facility j to customer i, y_j indicates whether facility j is established ($y_j = 1$) or not ($y_j = 0$). The constraint (5) makes sure that all demands have been met by the open sites, and the constraint (6) is to keep integrity. Since it is assumed that there is no capacity limit for any facility, the demand size of each customer is ignored, and therefore constraint (5) established without considering demand variable ($b_i = 1$).

Using this model, what has to be decided is (a) the number of facility sites to be established and (b) the quantities to be supplied from facility j to customer i such that the total cost (including fixed and variable costs) is minimised. UFL problems are usually represented in a binary way in which the facilities are denoted by 1 if opened and by 0 if closed. The representation of a solution in the UFL problem is a binary string with N bits, where N is the number of facilities of the problem.

A set of fifteen (Cap) benchmark problems that are accessible on OR Library [3] are tackled. In additions to these problems, three randomly generated (MQ) benchmark problems

Table 1 The benchmark problems and their sizes

Benchmark Problem Size	Cap71–Cap74 16×50	Cap101–Cap104 25×50	Cap131–Cap134 50×50	CapA–CapC 100×1000	MQ1–MQ3 500×500
Computational effort used for estimating the main effect	640	1,000	2,000	4,000	20,000

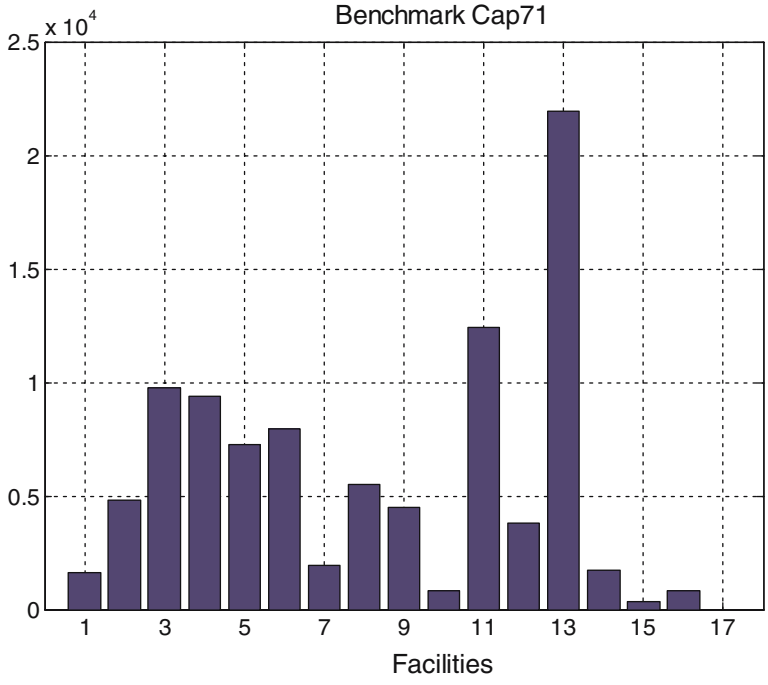

Fig. 4 Main effects of facilities of problem Cap71

are produced based on the method used by [12]. The benchmarks and their size are shown in the first and the second row of Table 6.1 respectively. The size of the problems indicates the number of customers and the number of facilities. For example, the size of the first four benchmarks (Cap71–Cap74) are 16 × 50 that means they are the problems with 16 facilities and 50 customers.

Equation (3) in Sect. 2 is used to estimate the main effect of each facility of the benchmarks. The number of function evaluations used to estimate the main effect in each facility is set to 40. Since they have 16 facilities, the total number of function evaluations used to estimate the main effects for all facilities in Cap71–Cap74 is 640 (i.e.: 16 × 40). Cap101, Cap131 and CapA have 25, 50 and 100 facilities in the problem respectively. MQ1 has 500 facilities in the problem. Therefore the total number of function evaluations used to estimate the main effects for all facilities in Cap101–Cap104, Cap131–Cap134, and CapA–CapC are 1,000 (i.e.: 25 × 40), 2,000 (i.e.: 50 × 40), and 4,000 (i.e.: 100 × 40) respectively. The total numbers of function evaluations used to estimate the main effects for all facilities in MQ1–MQ3 is 20,000 (i.e.: 40 × 500). The computational effort used for estimating the main effects for all facilities of the benchmarks is summarized in the third row of Table 1.

The main effects of facilities in the benchmarks of Cap71, Cap101, Cap131, CapA and MQ1 are shown in Figs. 4, 5, 6, 7 and 8 respectively. The y-axis shows the main effect of each facility and the x-axis shows the facilities. The main effect indicates the contribution of each facility to the total cost given by Eq. (4).

It can be seen from Fig. 1 that facilities in Cap 71 have different main effects. The distribution of main effects of facilities does not have a special sequence. Similarly Figs. 2, 3 and 4 also show the similar situation on Cap 101, Cap 131 and CapA respectively. Therefore the values of main effects of facilities of these Cap problems do not have a special sequence. On the other hand, Figure 5 shows that the main effect of the first facility in MQ1 has the highest main effect. The main effect of the last bit has the smallest main effect. With higher facility number, the main effect of the facility is lower.

Therefore, the figures show that in these functions, different facilities have different main effects, and thus each facility has different contribution to the cost function in Eq. (4). The

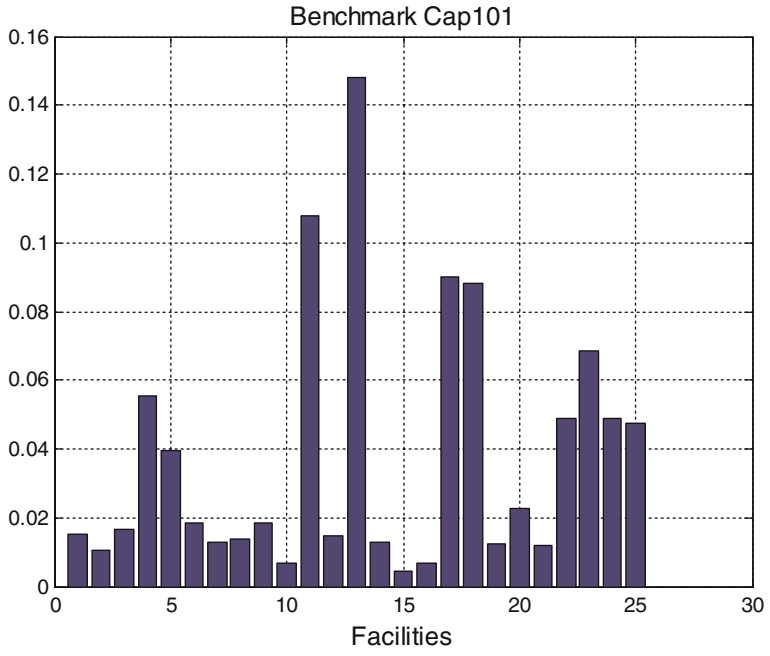

Fig. 5 Main effects of facilities of problem Cap101

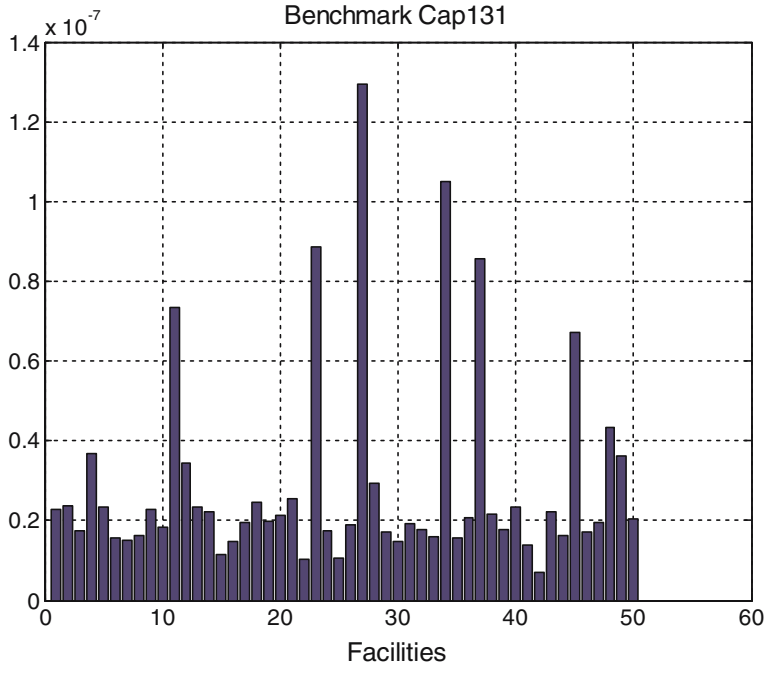

Fig. 6 Main effects of facilities of problem Cap131

cost value in Eq. (4) does not change so much when the facilities with low main effects are closed or opened. At the other extreme, the cost value changes a lot when the facilities with high main effects are closed or opened.

In the following sections, it is shown how to integrate this heuristic information of main effects into the mutation operator in the GA and the neighborhood function in the SA to help the search for the optimal solutions of UFL problems.

4 A fine-tuned mutation operator of GA

The idea of fine-tuning can be extended to the mutation operator for solving the UFL problems if the main effect in each facility is known. Normally the mutation rate is set to a constant magnitude across bit representation. Fogarty [10] demonstrated that changing probability of mutation across bit representation can improve the performance of GA. His

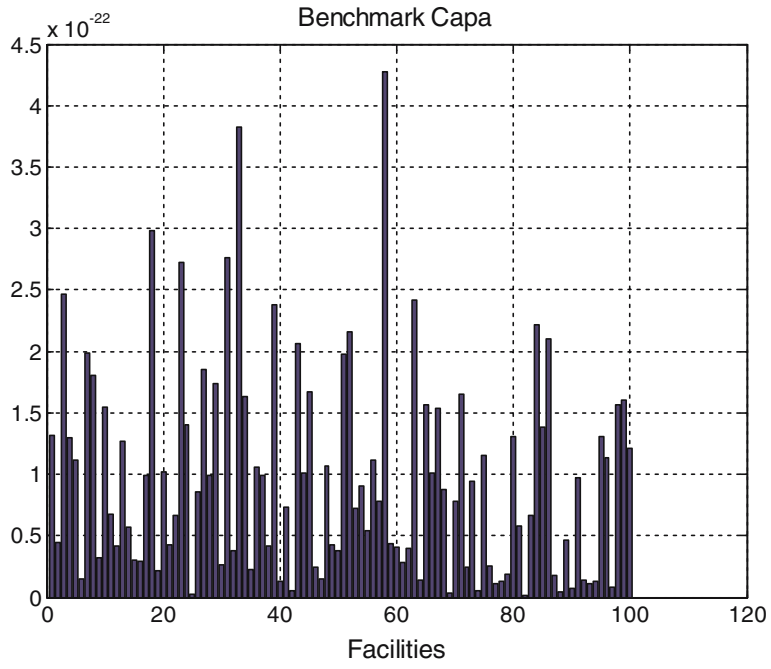

Fig. 7 Main effects of facilities of problem CapA

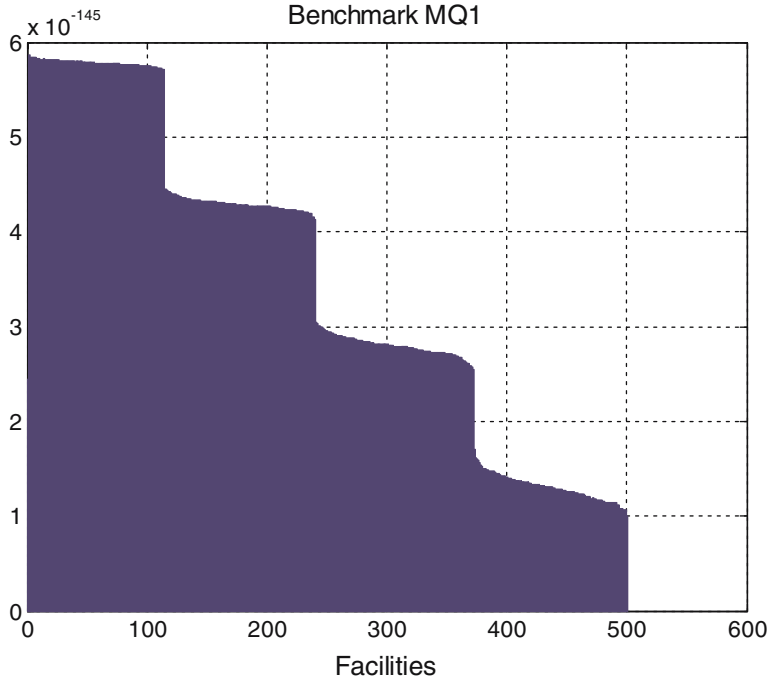

Fig. 8 Main effects of facilities of problem MQ1

approach is that the GA has a relatively higher chance to be fine-tuned for a better solution in the later generations, when more mutation is performed on bits with relatively lower main effect. Therefore, in the later generations, bits with relatively lower main effect should be mutated more frequently than those with relatively higher main effect.

For solving some combinatorial problems [Beasley and Chu 1994, [4]], it has been shown that a higher mutation rate is necessary when the population are converging. It has been also found that it is beneficial to utilise a variable mutation rate rather than a fixed one. In the earlier generations, the mutation rate is set to a low value to allow minimal

disruption, since mutation is not really needed. As the GA is progressing, the mutation rate increases. When the GA finally converges, the mutation rate becomes stable at a constant value. Therefore in the later generations new alleles can still be produced by performing mutation on genes. It leads that the GA can still have chances to keep progressing in the later generations. However, in this approach the mutation rate in each bit along the bit representation is identical.

In this paper we propose a more comprehensive approach for solving the UFL problems, which integrates the information of main effects in each bit of the genes. The new mutation regime with different mutation rate across the bit representation works as follows:

The similar approach used for solving the parametrical problems [5] and the combinatorial problems [6] is used to solve the UFL problems. The mutation rate of the bit with relatively lower main effect (less than $\mu - 3\sigma$) is defined as $P_{\max}(g)$ [as shown in Eq. (7)], where its dynamic behaviour is similar to that of [Beasley and Chu 94]. μ and σ is the mean and the standard deviation of the main effects of the bits respectively. To fine-tune the search in the later generations, the bit with relatively higher main effect (larger than $\mu + 3\sigma$) does not need to be mutated as frequently as the one with relatively lower main effect. Therefore the mutation rate of the bit with relatively higher main effect is defined to be $P_{\min}(g)$ [as shown in Eq. (8)], which is half of the value of $P_{\max}(g)$. The two parameters are defined respectively as:

$$P_{\max}(g) = \frac{m_f}{1 + \exp\left(\frac{-0.04 \times m_g (g - m_c)}{m_t}\right)} \tag{7}$$

and

$$P_{\min}(g) = 0.5 \times P_{\max}(g), \tag{8}$$

where g is the number of generations, m_f specifies the final stable mutation rate, m_c specifies the number of generations at which a mutation rate of $m_f/2$ is reached and m_g specifies the gradient at $g = m_c$.

For the rest of the bits main effects of which are between $\mu + 3\sigma$ and $\mu - 3\sigma$, their mutation rates are defined between the value of $P_{\max}(g)$ and $P_{\min}(g)$. In the early generations, both mutation rates of the bits with high or low main effects are set at a low value. In the later generations, the mutation rates of bits with low main effects are defined to be higher than the ones with high main effects. Therefore more mutations can be performed on bits with low main effects.

The mutation rate of the ith bit is given by the following rule:

$$P_i(g) \,|Mi,\, P\min(g),\, P_{\max}(g) = \begin{cases} P_{\min}(\mathrm{g}) & \text{if } M_i > \mu - 3\sigma \\ P_{\max}(\mathrm{g}) & \text{if } M_i \le \mu + 3\sigma \\ P_{\min}(g) & \\ -\frac{(P_{\max}(g) - P_{\min}(g)) \cdot [M_i - (\mu - 3\sigma)]}{6\sigma} & \text{if } \mu + 3\sigma > M_i \ge \mu - 3\sigma \end{cases} \tag{9}$$

where M_i is the main effect of the ith variable with $i = 1, 2, \ldots, n$. μ and σ are the mean and the standard deviation of the main effects in variables respectively. $P_{\min}(g)$ and $P_{\max}(g)$ are defined in Eqs. (7) and (8) respectively.

As an example, the mutation rates of bits of representation problem MQ1 is discussed. The main effects of bits of representation problem MQ1 has been shown in Fig. 8. It can be seen from Fig. 8 that the main effect of the first bit has the highest main effect. The main effect of the last bit has the smallest main effect. With higher bit number, the main effect of the bit is lower. It can also be seen from Fig. 8 that the main effects of the bits can be classified into four main classes – the class of bits with the highest main effect, the class of bits with the higher main effect, the class of bits with the lower main effect and the class of bits with the lowest main effect. The mutation rate of each bit is defined by Eq. (9) according to its main effect.

Figure 9 illustrates the mutation rates of bits of representation problem MQ3. The parameter of $P_{\max}(g)$ in Eq. (7) is set as follows: m_f is set to be 0.05, m_c is set to be 2/5, m_g is set to be $\pi/30$ and the total number of generations is set to be 200. The y-axis of Figure 9 shows the mutation rates of the bits. It shows that the mutation rates of the bits are set at zero in the early generations. They begin to increase their value at generation 60 and then saturate at generation 80.

The mutation rates of the bits are assigned according to their main effects. Figure 9 shows that the bits in the highest main effect class are assigned to the lowest mutation rate class. The bits in the higher, the lower and the lowest main effect class are assigned to the lower, higher and highest mutation rate class respectively. Therefore, in the later generations, it can be seen clearly that the bits with low main effects are assigned with high mutation rates, and the bits with high main effects are assigned with low mutation rates. The bits with low main effects are mutated more frequently than the ones with high main effects. The approach of fine-tuning is applied by mutating more frequently on the bits with low main effects.

Figure 10 shows the proposed mutation rates of the bits over the generations in a three-dimensional graph. The z-axis shows the mutation rates. It shows clearly that the mutation rates not only vary over the generations but also vary according to the main effects across the bit representation. The mutation rate of each bit is set according to its main effect. However, in classical GA, the mutation rates of bits are held constant throughout a run. At the same time, the mutation rates across bit representation are also kept constant. In Sect. 6, the empirical comparison between the proposed dynamic mutation rate and the constant mutation rate is discussed.

5 A fine-tuned neighbourhood function of SA

Same as the GA, UFL problems are represented in a binary way in the SA in which the open facilities are denoted by 1 and the closed ones by 0 [24]. The state of solutions is represented by a binary code, say B, denoting the enumerated open facility. For instance, $B = 0, 1, 0, 0, 1$ means the second and the

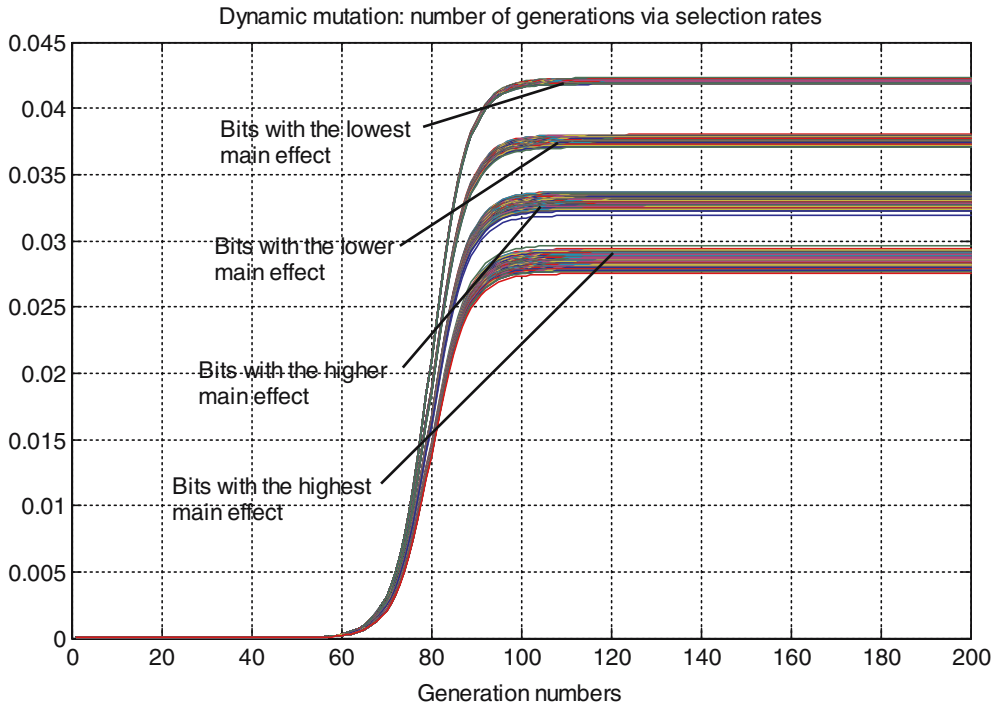

Fig. 9 A two-dimensional illustrtion of mutation rate varying both over generations and across bit representation

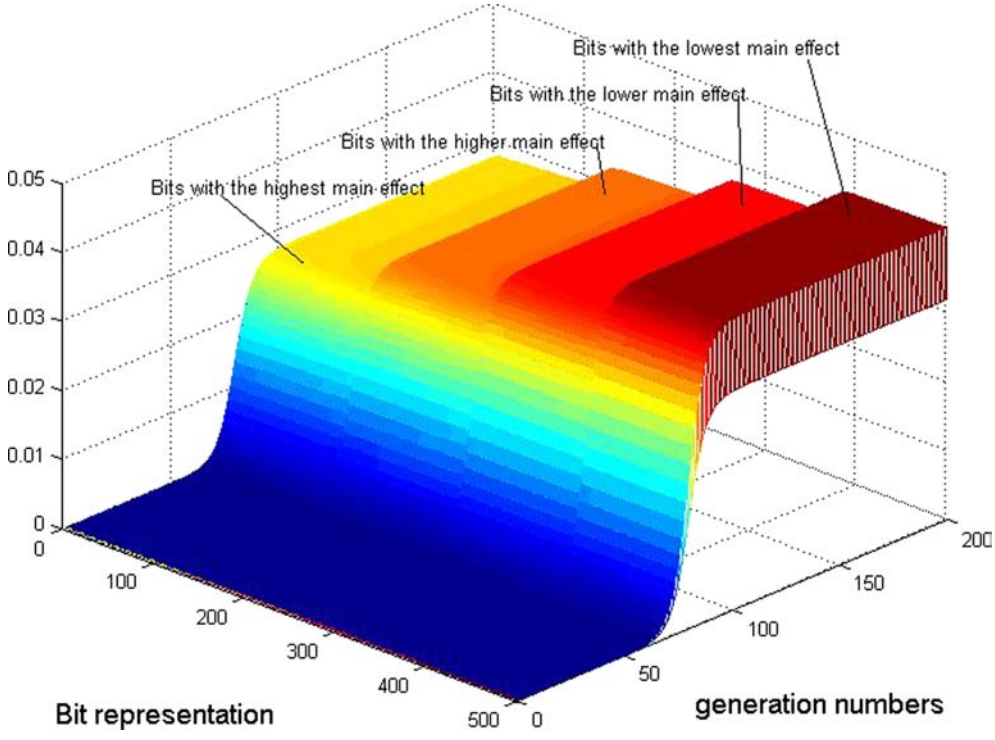

Fig. 10 A three-dimensional illustration of mutation rate varying both over generations and across bit representation

fifth facilities are opened, and the rest are closed. In order to move from one solution to another, the digits are briefly converted to each other. Depending on the heuristics employed, this conversion is done by a neighbourhood function.

The operation of neighbourhood function is similar to the mutation operation that mutates an individual bits (either from 0 to 1 or from 1 to 0) or exchanges states of two selected bits. Thus it employed allow us to modify the states by three

different operations: (1) exchange two digits on the bits; (2) change a digit from 0 to 1; (3) change a digit from 1 to 0, meaning exchange the state of one facility with another one, open a new facility or close one opened facility respectively. In the neighbourhood function, it selects the bit or bits randomly with equal chance in the representation. In this paper, the chance of selection of a bit is called the mutation rate. Usually the mutation rate of bits is set at a constant value $1/N$, where N is the length of the representation. The mutation rate is held constant throughout a run of the SA giving the distribution of the mutation rate over iterations. At the same time, the mutation rate is also kept constant across bit representation. A new approach of mutation rate, which is integrating with the information of main effect, is discussed follows:

The idea of using the information of the main effects for fine-tuning can be used to modify the mutation rate in the neighbourhood function of the SA. Similar to GAs, in later generations, the neighbourhood function can be used to fine-tune the searching process by mutating more frequently on the bits with low main effects.

The mutation rates of all bits are set at a constant value in the earlier generations. In the later generations, the mutation rates of the bits with low main effects increase exponentially to a value, and then saturate. At the same time, the mutation rates of the bits with high main effects decrease exponentially to a value, and then saturate. Therefore more bits with lower main effects are selected to be mutated in the later stages of the search.

Two parameters $P'_{\max}(g)$ and $P'_{\min}(g)$ are defined respectively as:

$$P'_{\min}(g) = m_f - P_{\max}(g) \tag{10}$$

and

$$P'_{\max}(g) = m_f + P_{\max}(g), \tag{11}$$

where g is the number of generations. $P_{\max}(g)$ can be found in Eq. (7).

The mutation rates of the bits with relatively high main effects (larger than $\mu + 3\sigma$) are defined as $P'_{\min}(g)$ in Eq. (10). Their values decrease exponentially when the generation number is increasing. After decreased to a particular value, the values saturate. The ones with relatively low main effects (less than $\mu - 3\sigma$) are defined as $P'_{\max}(g)$ in Eq. (11). Their values increase exponentially when the generation number is increasing. After increased to a particular value, the values saturate.

The mutation rates for the rest of the bits, which their main effects are between $\mu+3\sigma$ and $\mu-3\sigma$, are also defined based on their main effects. Their values are defined between the value of $P'_{\max}(g)$ and $P'_{\min}(g)$. In the later generations, the bit with low main effect is assigned to have higher mutation rate than the one with high main effect. At the same time, the bit with high main effect is assigned to have lower mutation rate.

The mutation rate of the ith bit is defined by Eq. (9): i.e.: $P_i(g)|_{M_i, P'_{\max}(g), P'_{\min}(g)}$, where M_i is the main effect of the ith bit with $i = 1, 2, \ldots, n$. $P'_{\max}(g)$ and $P'_{\min}(g)$ can be found in Eqs. (10) and (11) respectively.

As an example, the mutation rates for problem MQ1 are discussed. Figure 11 illustrates the mutation rates of the bits of representation problem MQ1. The parameters used at $P_{\max}(g)$ s in Eqs. (10) and (11) are set as follows: m_f is set to be 0.05, m_c is set to be 2/5, m_g is set to be $\pi/30$ and the total number of generations is set to be 200. The y-axis of Fig. 11 shows the mutation rates.

Figure 11 shows that before 120 generations, the mutation rates of all bits are at a constant value which is 0.02. The bits with the lowest and lower main effect start to increase after 120 generations, and saturate at about 180 generations. The increasing rates of the bits in the lowest main effect class are higher than the ones in the lower main effect class. On the other hand, the bits in the highest and higher main effect class start to decrease after 120 generations and saturate at about 180 generations. The decreasing rates of the bits in the highest main effect class are higher than the ones in the higher main effect class.

Therefore, in the later generations, the bits with low main effects are mutated more frequently than the ones with high main effects. The bits with high main effects are not selected for mutation as frequently as the ones with low main effects. The approach of fine-tuning can be applied by mutating more frequently on the bits with low main effects.

Figure 12 shows the mutation rates of the bits over the generations in a three-dimensional graph. The z-axis shows the mutation rate. It shows that the proposed dynamic mutation rates not only vary over the generations but also vary according to the main effects across bit representation. The mutation rate of each bit is set according to its main effect. However, in the standard SA, the mutation rates in bits are held constant throughout a run and are also kept constant across bit representation. In Sect. 6, the empirical comparison between the proposed dynamic mutation rate and the constant mutation rate is discussed.

6 Experimental results

This section presents an analytic comparison for the proposed methods and shows a comparative work with very recent publications for the UFL problems: Kratica et al's GA [12] and Aydin and Fogarty's SA [Aydin and Fogarty 2004]. The algorithms are run for the 18 benchmark problems [3,12] shown in Table 1. The benchmark problems were tested using the following algorithms:

1) Kratica et al's [12] algorithm (KGA): KGA is proposed by Kratica et al. [12] for solving the UFL problems. The detailed steps of the algorithm are shown in the appendix. In this GA, the constant mutation rate of $1/N$ is used, where N is the number of bits of the representation.
2) Modified Kratica et al's algorithm (MKGA): the basic process of MKGA is identical to KGA except the mutation operator utilises the proposed dynamic mutation rate discussed in Sect. 4.

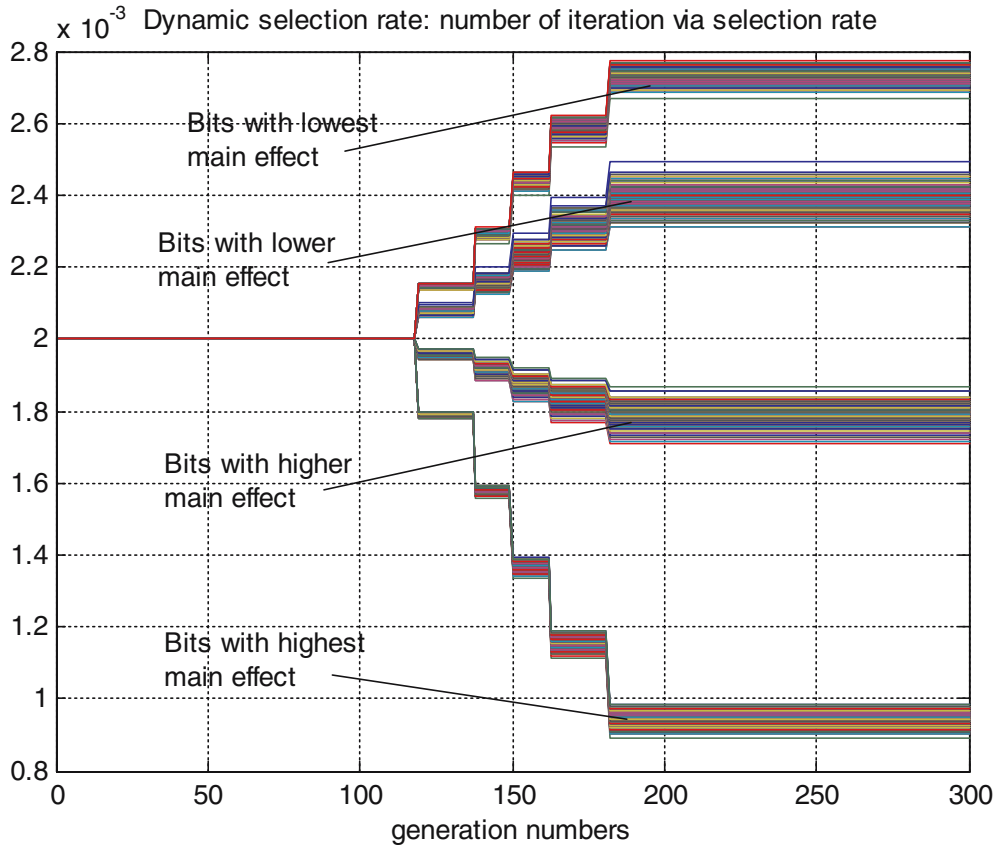

Fig. 11 A two-dimensional illustration of mutation rates varying both over generations and across bit representation

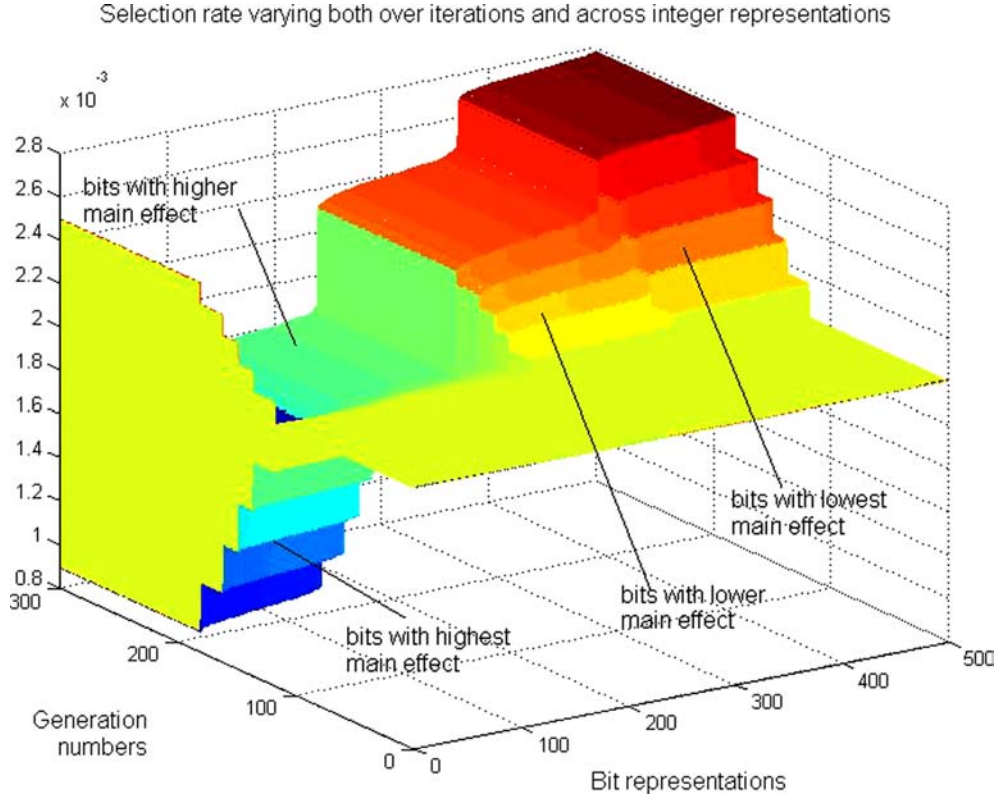

Fig. 12 A three-dimensional illustration of dynamic mutation rates varying both over generations and across bit representation

Table 2 The pre-defined number of evaluations used in KEA, ASA, MKEA and MASA

Benchmark	Cap71–74	Cap101–104	Cap131–134	CapA–C	MQ1–3
Computational effort used for estimating the main effect	640	1,000	2,000	4,000	20,000
Computational effort used in MKEA and MASA	40,000	100,000	200,000	200,000	200,000
Computational effort used in KEA and ASA	40,640	101,000	202,000	104,000	220,000

3) Aydin and Fogarty [2] simulated annealing (ASA): ASA is used by Yigit et al. [24] for solving the UFL problems. The detailed steps of the algorithm are shown in the appendix. A constant mutation rate of $1/N$ is used in the neighborhood function, where N is the number of bits of the representation.
4) Modified Aydin et al's simulated annealing (MASA): the basic process of MASA is identical to the ASA except the neighborhood function utilises the proposed dynamic mutation rate discussed in Sect. 5.

In order to have a fair comparison with the latest work, all the algorithms were re-encoded to run on a pentium-4 machine and using the same programming language Matlab. The following parameter values have been used for KGA and MKGA: crossover rate = 1.0 in both KGA and MKGA. Simple two-point crossover operation and rank based selection operation was used in both KGA and MKGA. Experiments are done 50 times for each benchmark.

In both MKGA and MASA computational effort has been used for estimating the main effect in each facility on the problems. To make a comparison fair, the computational effort used for estimating the main effect in each facility in MKGA and MASA is added to the total computational effort used in KGA and ASA, when evaluating the performance of the algorithms. The number of function evaluations used for estimating the main effect in each benchmark is shown in the second row of Table 2. The pre-defined number of function evaluations used in MKGA and MASA, and KGA and ASA are shown in the third and fourth rows of Table 2 respectively.

Table 3 presents the results yielded by the algorithms. The columns show the information in the following order: the name of the benchmark, its optimum value, the average and standard deviation of results found in KGA and MKGA, whether KGA and MKGA can find the optimal solution or not, the average and standard deviation of results found in ASA and MASA, and whether ASA and MASA can find the optimal solution or not. The optimal solutions of the problem MQ1, MQ2 and MQ3 are not known so we put 'N/A' in the column of the optimal solution.

Table 3 shows that MKGA can find the optimal solution of 9 of the 15 Cap problems for which the optimal solutions are known, but KGA can only find 7 of the 15 Cap problems. Furthermore, KGA cannot find the optimal solution of Cap102 and Cap134 but MKGA can find their optimal solution. On the other hand, it also shows that MASA can find the optimal solution of 12 of the 15 Cap problems but ASA can only find 11 of them. ASA cannot find the optimal solution of Cap 133 but MASA can.

For the problems that both MKEA and KEA cannot reach the optimal solutions, it can be seen from Table 3 that in general the average results found by MKEA is better than KEA. Furthermore, MKEA gives smaller standard deviations of results than KEA, and hence the solution quality of MKEA is more stable. On the other hand, for the problems that both MASA and ASA cannot reach the optimal solutions, it can be seen from Table 3 that MASA can find better average results than ASA. Furthermore MASA gives smaller standard deviations of results than ASA. Therefore, in both solution quality and solution stability, MKEA is better than KEA and MASA is better than ASA.

Then the t-test is employed to assess whether the performance between the algorithms is statistically different in the problems in which the algorithms cannot reach the optimum. For the problems that both KGA and MKGA cannot reach the optimal solutions, the t-values between KGA and MKGA are shown in Fig. 13. For the problems that both ASA and MASA cannot reach the optimal solutions, the t-values between ASA and MASA are shown in Fig. 14. When the t-value is more than 1.645, there is a significantly difference between the algorithms with 95% confidence level. It can be seen from Fig. 13 that the t-values in all the problems are higher than 1.645. Therefore, for the problems that both MKGA and KGA cannot reach the optimal solutions, the solutions found by MKGA are significantly better than the ones found by KGA. And, Fig. 14 also shows that all t-values between ASA and MASA are more than 1.645. Therefore MASA is significantly better than ASA for solving the problems that both SAs cannot reach the optimal solutions.

For the problems in which both KGA and MKGA can reach the optimal solutions, Table 4 shows the computational effort used by them to reach the optimal solutions. For KGA, the computational effort is based on the number of function evaluations performed to reach the optimum. For MKGA, the computational effort is based on the number of function evaluations performed to reach the optimum plus the number of functions evaluations taken to estimate the main effect. It can be seen from Table 4 that KGA is faster than MKGA only in the problem Cap 71. In the rest of the problems MKGA used less computational effort to reach the optimal solution than the one used by KGA. Therefore in general MKGA can find the optimal solution with less number of function evaluations than KGA in which the problems that both can reach the optimum.

On the other hand, Table 5 shows the computational effort used by ASA and MASA for the problems that both can reach the optimal solutions. For ASA, the computational effort is based on the number of function evaluations performed to reach the optimum. For MASA, the computational effort is based on the number of function evaluations performed to reach the optimum plus the number of function evaluations

Table 3 Experimental results obtained from the algorithms

Benchmark	Optimum	Genetic algorithms			Simulated annealings		
		KGA average of found results (standard deviation)	MKGA average of found results (standard deviation)	Algorithm that can reach the optimum	ASA average of found results (standard deviation)	MASA average of found results (standard deviation)	Algorithm that can reach the optimum
Cap 71	932615.75	932615.75 (0.00)	932615.75 (0.00)	KGA,MKGA	932615.75 (0.00)	932615.75 (0.00)	ASA,MASA
Cap 72	977799.40	977799.40 (0.00)	977799.40 (0.00)	KGA,MKGA	977799.40 (0.00)	977799.40 (0.00)	ASA,MASA
Cap 73	1010641.45	1010641.45 (0.00)	1010641.45 (0.00)	KGA,MKGA	1010641.45 (0.00)	1010641.45 (0.00)	ASA,MASA
Cap 74	1034976.98	1034976.98 (0.00)	1034976.98 (0.00)	KGA,MKGA	1034976.98 (0.00)	1034976.98 (0.00)	ASA,MASA
Cap 101	796648.44	796648.44 (0.00)	796648.44 (0.00)	KGA,MKGA	796648.44 (0.00)	796648.44 (0.00)	ASA,MASA
Cap 102	854704.20	854719.45 (107.85)	854704.20 (0.00)	MKGA	854704.20 (0.00)	854704.20 (0.00)	ASA,MASA
Cap 103	893782.11	893838.63 (137.52)	893811.07 (131.00)	None	893782.11 (0.00)	893782.11 (0.00)	ASA,MASA
Cap 104	928941.75	929158.17 (342.79)	928941.75 (0.00)	MKGA	928941.75 (0.00)	928941.75 (0.00)	ASA,MASA
Cap 131	793439.56	793514.35 (227.35)	793453.96 (101.79)	None	793512.76 (241.20)	793444.62 (87.56)	None
Cap 132	851495.33	851591.56 (215.42)	851516.67 (181.60)	None	851495.33 (0.00)	851495.33 (0.00)	ASA,MASA
Cap 133	893076.71	893508.04 (395.00)	893382.43 (369.39)	None	893080.21 (17.88)	893076.71 (0.00)	MASA
Cap 134	928941.75	928941.75 (0.00)	928941.75 (0.00)	KGA,MKGA	928941.75 (0.00)	928941.75 (0.00)	ASA,MASA
Cap A	17156454.48	17156454.48 (0.00)	17156454.48 (0.00)	KGA,MKGA	17156454.48 (0.00)	17156454.48 (0.00)	ASA,MASA
Cap B	12979071.58	13011678.12 (76981.17)	12990616.85 (29903.01)	None	12987614.99 (23838.05)	12978837.46 (22448.28)	None
Cap C	11505594.33	11537003.29 (31858.74)	11526188.73 (30726.15)	None	11549306.50 (26660.80)	11510569.44 (21555.32)	None
MQ1	N/A	2588.67 (29.4240)	2577.56 (29.1737)	None	2543.93 (12.8277)	2537.09 (12.7607)	None
MQ2	N/A	2557.95 (27.6521)	2548.35 (22.1426)	None	2508.30 (15.1940)	2502.66 (14.9211)	None
MQ3	N/A	2576.18 (27.5972)	2566.44 (23.8971)	None	2534.98 (13.6499)	2528.17 (13.5800)	None

Table 4 Number of function evaluations performed in KGA and MKGA until the optimal solution reached in the problems in which both algorithms can reach the optimum

Benchmark	Number of function evaluations performed in KGA	Number of function evaluations performed in MKGA + number of function evaluations taken to estimate the main effect
Cap 71	5,576	6,216
Cap 72	9,492	9,196
Cap 73	8,982	8,760
Cap 74	5,180	4,876
Cap 101	16,220	15,240
Cap 134	26,644	25,768
Cap A	234,892	196,828

taken to estimate the main effect. It shows that ASA is faster than MASA only in the problem Cap 71 and Cap 72, but in the rest of the problems it shows that MASA can find the optimal solution with less number of function evaluations than the one used on ASA. Therefore, ASA is better than MASA in only two of the ten problems, and the problem sizes of the two problems are the smallest in the benchmarks. However, MASA is faster than ASA in eight of the ten problems.

In the two tested GAs (i.e.: KGA and MKGA), their steps are identical except the mutation rates used. The constant mutation rate is used in KGA and the proposed dynamic mutation rate (discussed in Sect. 4) is used in MKGA. On the other hand, in the two tested SAs (i.e.: ASA and MASA), their steps are also identical except the mutation rate used in the neighbourhood function. In ASA, the constant mutation rate is used. In MASA, the proposed dynamic mutation rate

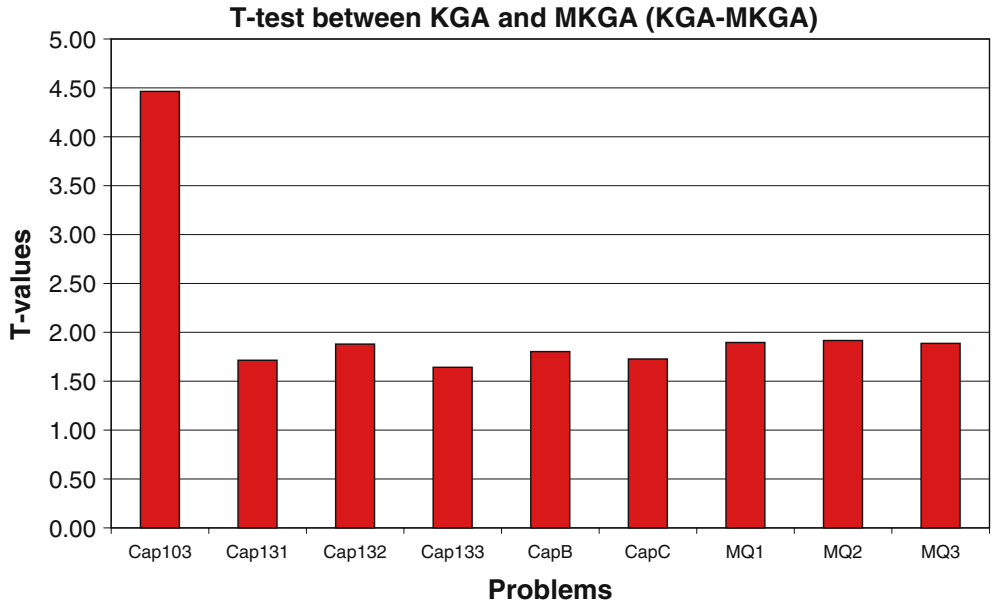

Fig. 13 *T*-value between KGA and MKGA (KGA-MKGA) on the problems that both cannot reach the optimum

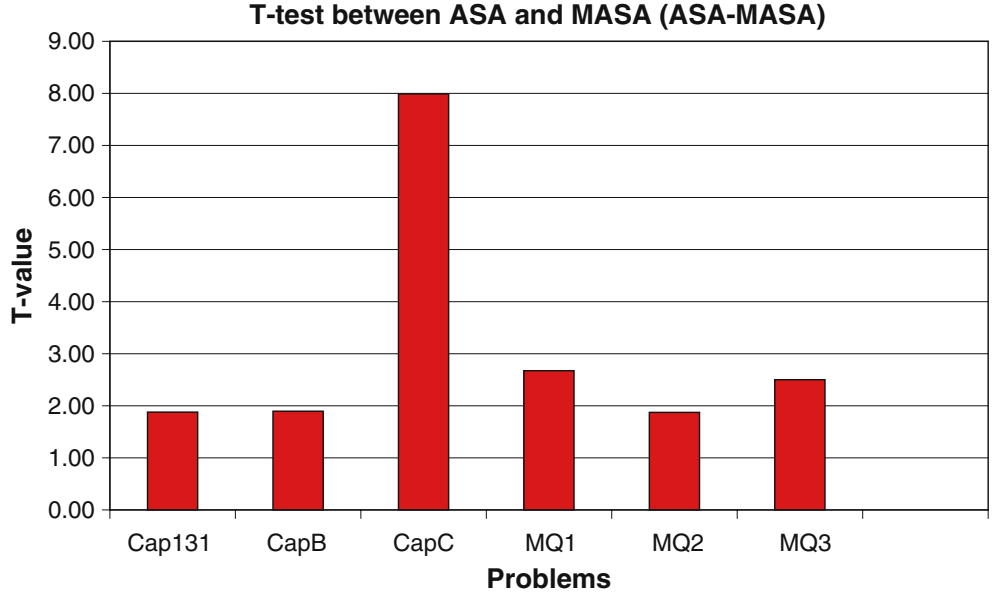

Fig. 14 *T*-value between ASA and MASA (ASA–MASA) on the problems that both cannot reach the optimum

(discussed in Sect. 5) is used. These results indicate that both proposed approaches aid to find the optimal solutions in these benchmark problems.

7 Conclusion

In this paper, we have shown how to estimate main effects of gene representation. We have introduced the notion of how to use the main effects of genes to fine-tune the searching process in the stochastic searhing methods GA and SA. A new approach of mutation rate in the GA and slection rate in the SA have been proposed by integrating the information of main effects of gene representation. As an illustration we used the proposed methods to solve the UFL problem. Then we compared the performance of the proposed methods with the most recent existing methods for solving this problem. The experimental results show that the proposed methods outperform the existing methods. They aid to search for better solutions when the existing methods are not able to reach

Table 5 Number of function evaluations performed in ASA and MASA until the optimal solution reached in the problems in which both algorithms can reach the optimum

Benchmark	Number of function evaluations performed in ASA	Number of function evaluations performed in MASA + number of function evaluations taken to estimate the main effect
Cap 71	600	1,040
Cap 72	600	1,040
Cap 73	2,800	2,440
Cap 74	600	1,040
Cap 101	7,000	7,200
Cap 103	25,600	16,400
Cap 104	1,600	1,400
Cap 132	9,400	8,000
Cap 134	18,00	3,600
Cap A	15,800	12,200

the optimal values. In addition, they use less computational effort to reach the optimal solutions for the problems that both the proposed and the existing methods are able to reach the optimal solutions.

We are currently working on an extension of this technique to solve the nonbinary problems such as the assignment problem. Furthermore, we are also improving this technique by considering the notion of the ruggedness of the fitness landscapes of the problems.

Appendix

The algorithm of Kratica's Genetic Algorithm (KGA):

Begin
$i = 1$
Initialise the population $P(i)$
Repeat
Evaluate the fitness of the individuals: $f(i) = eval(P(i))$
$P(i+1) = selection\ (P(i), f(i))$
$P(i+1) = crossover\ (P(i+1))$
$P(i+1) = mutation\ (P(i+1))$
$i = i + 1$
Until $i =$ *pre-defined number of generation*
End.

KGA [12] is similar to the classical GA. First, a population of solutions is randomly initialised, and then the number of generations is set. After that, each individual of the population is assigned a fitness value according to the quality of the solution. Then a new population is formed by selecting the better fitted individuals to reproduce more often than the worst ones. And then the population evolves towards better solutions by crossover and mutation. For selection, the rank based selection mechanism is used. For crossover, the uniform crossover introduced by Syswerda [18] is used. For mutation, a simple bit-by-bit mutation is used.

The algorithm of Aydin's Simulated Annealing (ASA):

Begin
$i = 1$
Initialise the population $P(i)$
Repeat
$t = 100$
Pick one s_t *from* $P(i)$
Repeat
$s_{0.955t}$=*neighbour_function*(s_t)
evaluate the fitness of $s_{0.955t}$ *and*
s_t: $f_{0.955t} = fun(s_{0.955t})$
and $f_t = fun(s_t)$
If $f_{0.955t} < f_t$ *then replace* s_t *with* $s_{0.955t}$
Else
generate a random number r
If $exp(-(f_{0.955t} - f_t)/t) > r$ *then replace*
s_t *with* $s_{0.955t}$
Endif
Endif
$t = t * 0.955$
Until $t < 0.01$
$i = i + 1$
Put s_t *back into the population* $P(i)$
Until $i =$ *(pre-defined number of generations)*
End.

ASA [1,2,24] is implemented to evolve a population of solutions with an ASA operator up to predefined number of generations. First, a population of solutions is randomly initialised, and then the number of generations is set. After that, a particular solution is randomly selected for being operated by an ASA operator. The ASA operator starts with the highest temperature (t=100). It modifies the randomly selected solution by using the neighborhood function and judges if the yielded solution will be promoted for the next inner iteration. Then it cools the temperature by a cooling coefficient (0.955) at the end of each inner iteration. When the temperature cools to 0.01, ASA operator finishes with 200 inner iterations, and releases the solution. The solution obtained through this process is put back into the population subject to a replacement rule. If the replacement rule is satisfied, then the new solution is replaced with the old one. That is the end of one generation. This process repeats until the pre-defined number of generations is completed.

Acknowledgements The first author is grateful to express his sincere thanks to Luis Hercog and James Werner for many useful discussions and valuable suggestions.

References

1. Aydin ME, Fogarty TC (2001) Simulated annealing with evolutionary process for job-shop scheduling problems. In: EUROGEN 2001 – Evolutionary methods for design, optimisation and control with applications to industrial problems, 19–21 September 2001, Athens, Greece

2. Aydin ME, Fogarty TC (2004) A distributed evolutionary simulated annealing for combinatorial optimisation problems. J Heuristics 10:3 (accepted)
3. Beasley JE (1996) Obtaining test problems via Internet. J Glob Optim 8:429–433, http://mscmga.ms.ic.ac.uk/info.html
4. Beasley JE, Chu PC (1994) A genetic algorithm for the set covering problem. Eur J Oper Res 94:392–404
5. Chan KY, Aydin ME, Fogarty TC (2003) An epistasis measure based on the analysis of variance for the real-coded representation in genetic algorithm. In: Proceedings of the IEEE international congress on evolutionary computation, pp. 297–304
6. Chan KY, Aydin ME, Fogarty TC (2004) Parameterisation of mutation in evolutionary algorithm using the estimated main effect of genes. In: Proceedings of the IEEE international congress on evolutionary computation
7. Chatterjee S, Carrera C, Lynch LA (1995) Genetic algorithms and traveling salesman problems. Eur J Oper Res 93:490–510
8. Chu PC, Beasley JE (1997) A genetic algorithm for the generalised assignment problem. Comput Oper Res 24(1):17–23
9. Daskin MS (1995) Network and discrete location: models, algorithms, and applications. Wiley, New York
10. Fogarty TC (1989) Varying the probability of mutation in the genetic algorithm. In: Proceedings of the 3rd international conferences on genetic algorithms pp 104–109
11. Kirkpatrick S, Gelatt CD, JR., Vecchi MP (1983) Optimization by simulated annealing. Sci 220:671–680
12. Kratica J, Tosic D, Filipovic V, Ljubic I (2001) Solving the simple plant location problem by genetic algorithm. RAIRO Oper Res 35:127–142
13. Lin CKY, Haley KB, Sparks C (1995) A comparative study of both standard and adaptive versions of threshold accepting and simulated annealing algorithms in three scheduling problems. Eur J Oper Res 83:330–346
14. Montgomery DC (1997) Design and analysis of experiments. Wiley, New York
15. Reeves C (1994) Genetic algorithms and neighbourhood search. In: Evolutionary computing: AISB workshop, pp 115–130
16. Reeves C (1997) Genetic algorithms for operations researcher. Informs J Comput 9(3):231–250
17. Ruiz-Torres AJ, Enscore EE, Barton RR (1997) Simulated annealing heuristics for the average flow-time and the number of tardy jobs bi-criteria identical parallel machine problem. Comput Ind Eng 33(1–2):257–260
18. Syswerda G (1989) Uniform crossover in genetic algorithms. In: Proceeding of the 3rd international conference on genetic algorithms, pp 2–9
19. Szu H, Hartley R (1987) Nonconvex optimization by fast simulated annealing. Proc IEEE 75(11):1538–1540
20. Vaessens RJM, Aarts EHL, Lenstra JK (1992) A local search template. In: Proceedings of parallel problem-solving from nature 2. pp 65–74
21. Wilson JM (1997) A genetic algorithm for the generalised assignment problem. J Oper Res Soc 48:804–809
22. Yao X (1991) Simulated annealing with extended neighbourhood. Int J Comput Math 40:169–189
23. Yao X (1993) Comparison of different neighbourhood sizes in simulated annealing. In: Proceedings of 4th Australian conference on neural networks, pp 216–219
24. Yigit V, Aydin ME, Turkbey O (2004) Evolutionary simulated annealing algorithms for uncapacitated facility location problems. In: Proceedings of adaptive computing in design and manufacture, pp 20–22