Ant Colony Optimization with Immigrants Schemes in Dynamic Environments

Michalis Mavrovouniotis[1] and Shengxiang Yang[2]

[1] Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, United Kingdom
mm251@mcs.le.ac.uk
[2] Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex UB8 3PH, United Kingdom
shengxiang.yang@brunel.ac.uk

Abstract. In recent years, there has been a growing interest in addressing dynamic optimization problems (DOPs) using evolutionary algorithms (EAs). Several approaches have been developed for EAs to increase the diversity of the population and enhance the performance of the algorithm for DOPs. Among these approaches, immigrants schemes have been found beneficial for EAs in DOPs. In this paper, random, elitism-based, and hybrid immigrants schemes are applied to ant colony optimization (ACO) for the dynamic travelling salesman problem (DTSP). The experimental results show that random immigrants are beneficial for ACO in fast changing environments, whereas elitism-based immigrants are beneficial for ACO in slowly changing environments. The ACO algorithm with hybrid immigrants scheme combines the merits of the random and elitism-based immigrants schemes. Moreover, the results show that the proposed algorithms outperform compared approaches in almost all dynamic test cases and that immigrant schemes efficiently improve the performance of ACO algorithms in DTSP.

Key words: Ant Colony Optimization, Immigrants Schemes, Dynamic Optimization

1 Introduction

Ant colony optimization (ACO) algorithms emulate the behaviour of real ant colonies when they search for food from their nest to food sources. Ants communicate using their pheromone trails in order to complete this task as efficiently as possible. ACO algorithms have proved to be able to solve different optimization problems in real-world applications [2, 3]. Traditionally, researchers have been focused on stationary optimization problems, where their environment remains fixed during the execution of the algorithm. However, many real-world applications have dynamic environments. The problem then becomes more challenging since the optimum needs to be tracked when dynamic changes occur [12].

Traditional ACO algorithms have been designed for stationary optimization problems [3], and may not be sufficient anymore for DOPs. This is due to the fact

that the pheromone trails of the previous environment will not make sense for a new environment, after a change occurs. A simple way to address this problem is to re-initialize the pheromone trails and consider every change as the arrival of a new problem instance which needs to be solved from scratch. Unfortunately, this restart strategy is computationally expensive and usually not efficient.

Recently, developing ACO algorithms for DOPs has attracted a lot of attention since they can be useful for real-world applications. Thus, more specialized strategies have been proposed to maintain the high quality of output efficiently, which include local and global restart strategies [8], pheromone manipulation schemes to maintain or increase diversity [4], and memory-based approaches [6, 9]. These methods have been applied on the dynamic travelling salesman problem (DTSP) due to its importance for many real-world applications. One of the most efficient and well-studied methods is the memory-based version of ACO, known as the population-based ACO (P-ACO) algorithm [7]. It has a different framework from a traditional ACO algorithm since it maintains a population list (memory), which stores the best ant of every iteration, and is used to generate the pheromone trails. Taking a closer look at P-ACO, we see that it has the characteristics of a genetic algorithm (GA) [11] because of the memory. Thus, it inherits the disadvantage of a GA when a dynamic change may affect the individual on the genotypic level, which needs to be repaired. Often, the repair procedure is computationally expensive.

As we have seen on many GAs, immigrants schemes are advantageous when applied to DOPs [14, 15, 17]. Immigrants schemes enable the algorithm to maintain the diversity of the population to a certain level, by introducing new individuals into the current population. In this paper, we apply immigrants schemes into P-ACO. However, instead of using a long-term memory as in P-ACO, we use a short-term memory, where all the new ants replace the old ones to form a new population. Later on, a percentage of the worst ants are replaced by immigrants. We introduce three types of immigrants, which are traditional random, elitism-based, and hybrid immigrants. The experimental results show that immigrant schemes enhance the performance of ACO into DOPs. However, different immigrants schemes are advantageous under different environmental conditions.

The rest of the paper is organized as follows. Section 2 describes the standard ACO and P-ACO algorithms. Moreover, it describes how they are applied to the DTSP. Section 3 describes our proposed approaches where we apply immigrants schemes into P-ACO. Section 4 describes the experiments carried out by comparing our proposed approaches with P-ACO. Finally, Section 5 concludes this paper with directions for future work.

2 ACO for Dynamic Environments

2.1 Standard ACO

The traditional ACO algorithm consists of a population of μ ants, where each ant consists of two modes, the forward mode and the backward mode. Initially, all

ants are placed on a randomly selected city for a TSP and all pheromone trails are initialized with an equal amount of pheromone. All ants on their forward mode choose the next city based on pheromones and some heuristic information using a probabilistic decision rule, which is defined as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \text{if } j \in N_i^k \tag{1}$$

where τ_{ij} is the existing pheromone trail between city i and city j, η_{ij} is the heuristic information available a priory, which is defined as $1/d_{ij}$ and d_{ij} is the distance between the cities. N_i^k denotes the neighbourhood of cities of ant k when being on city i. α and β are the two parameters that determine the relative influence of pheromone trail and heuristic information, respectively.

Later on, all ants proceed to their backward mode by retracing their solutions and deposit pheromone according to their solution quality on the corresponding trails. However, before adding any pheromone, a constant amount of pheromone is deduced from all trails due to the pheromone evaporation, which is defined as:

$$\tau_{ij} \leftarrow (1 - \rho)\, \tau_{ij}, \forall\ (i, j), \tag{2}$$

where $0 < \rho \leq 1$ is the rate of evaporation. Reducing the pheromone values enables the algorithm to forget bad decisions made in previous iterations [3]. After evaporation, all ants deposit pheromone to the corresponding trails of their tour as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^k, \forall\ (i, j) \in T^k, \tag{3}$$

where $\Delta\tau_{ij}^k = 1/C^k$ is the amount of pheromone that ant k deposits and C^k is the cost of the tour T^k.

2.2 Population-Based ACO

The P-ACO algorithm is the memory-based version of an ACO algorithm, which was first applied on the stationary TSP [7]. It differs from the standard ACO algorithm described above since it follows a different framework. Generally, the algorithm maintains a population of solutions, which is used to update pheromone trails without any evaporation.

The initial phase and the first iterations of the P-ACO algorithm work in the same way as with the standard ACO algorithm. The pheromone trails are initialized with an equal amount of pheromone and the population list of a size K is empty. For the first K iterations, the iteration best ant deposits a constant amount of pheromone using Eq. (3) where $\Delta\tau_{ij}^k = (\tau_{max} - \tau_{init})/K$. Here, τ_{max} and τ_{init} denote the maximum and initial pheromone amount, respectively. This positive update procedure is performed whenever an ant enters the population list. On iteration $K+1$, the ant that has entered the population list first needs to

be removed in order to make room for the new one, and thus, a negative update to its pheromone trails is done, as follows:

$$\tau_{ij} \leftarrow \tau_{ij} - \Delta\tau_{ij}^{k}, \forall\ (i,j) \in T^{k}, \tag{4}$$

where $\Delta\tau_{ij}^{k}$ is defined as in the positive update above.

This strategy is based on the *Age* of ants. However, other strategies have also been proposed by researchers, such as *Quality* and *Prob* [6]. From the experimental results in [6], the default *Age* strategy is more consistent and performs better than the others, since the other strategies have more chances to maintain identical ants into the population list, which leads the algorithm to the stagnation behaviour. This is due to the fact that high levels of pheromone will be generated into a single trail and dominates the search space. Moreover, we have seen the importance of keeping the pheromone trails into a certain level from the Max-Min Ant System (MMAS) [13], which is one of the state-of-the-art ACO algorithms on stationary problems.

2.3 Response to Dynamic Changes

Theoretically, ACO algorithms can adapt to dynamic changes since they are inspired from nature, which is a continuous changing environment [12]. In practice, they can adapt by transferring knowledge from past environments [1]. So far, the description of ACO algorithms above has been made assuming stationary environments. Considering the DTSP, ACO needs to be modified in order to adapt to environmental changes efficiently.

The dynamics of adding/deleting a city affects both the genotypic and, usually, the phenotypic level of the ant. Therefore, considering that the solutions are affected by the change in iteration n, the pheromone trails will not make sense in iteration $n+1$. For the ACO algorithms that follow the traditional framework, it is vital to re-initialize the pheromone trails after a dynamic change, which acts as a restart of the algorithm.

For the P-ACO approach, the solutions stored in the population list are repaired and the pheromone trails are re-generated accordingly. This strategy is called *KeepElitst* [9] and uses two greedy heuristics to repair the genotype of the population: 1) the offended cities are removed from the solutions; and 2) the new cities are placed individually in a greedy fashion where they cause the minimum increase on the phenotype.

3 Incorporating Immigrants Schemes to ACO Algorithms

When addressing DOPs, traditional ACO algorithms cannot adapt well to the environmental changes once the ants reach the stagnation behaviour, where they follow the same path. The algorithm loses its adaption capability since it does not maintain diversity within the population. A good start has been made with the P-ACO algorithm with the use of memory, which maintains a certain level of

diversity and enables ACO algorithms to be more efficient for DOPs. However, it is a long-term memory and the solutions need to be repaired once a city is added or removed. Usually, the repair procedures requires prior knowledge of the problem and is computationally expensive.

As mentioned above, the application of immigrants schemes has been found efficient for GAs for DOPs. The principle is to introduce new individuals into the current population by replacing a percentage of individuals in the population [5]. The percentage should be relatively small because a high percentage may lead the algorithm into a too high level of diversity. High diversity does not always mean good performance on DOPs, because it may lead the algorithm into randomization [15, 17].

In this paper, we apply immigrants schemes into the P-ACO algorithm to maintain a certain level of diversity in the population and enhance its dynamic performance. However, we use a short-term memory where the ants of the current iteration replace the ants of the old iteration. Moreover, a percentage of immigrants replace the worst ants of the current population.

The advantages of using a short-term memory are closely related to the survival of ants in a dynamic environment, where no ant can survive in more than one iteration. This way, there is no need to use any repair algorithm (apart from the best ant of the previous iteration for the elitism-based immigrant scheme) because the changes do not affect the ants stored. Furthermore, there is one main concern that involves immigrants schemes, i.e., how to generate immigrants.

3.1 Random Immigrants ACO

The random immigrants ACO (RIACO) algorithm uses an immigrants scheme where ants are generated randomly, and replace the worst ones of the current population stored in the short-term memory every iteration. It is believed that "the continuous adaption of such algorithms makes sense only when the environmental changes of a problem are small to medium" [12]. This is due to the fact that the old environment has more chance to be similar with the new one. After a change occurs, transferring knowledge from the old environment may provide a good solution efficiently.

Considering this argument, RIACO may be suitable when changes are not slight since it provides diversity without considering any knowledge from the old environment. Moreover, it may be suitable in fast changing environments where information from the past may not be useful, since the algorithm does not have sufficient time to converge onto a good solution in order to gain knowledge.

3.2 Elitism-Based Immigrants ACO

The elitism-based immigrants ACO (EIACO) algorithm uses an immigrants scheme where ants are generated by mutating the best ant of the previous iteration. These immigrants also replace the worst ones in the short-term memory every iteration as in RIACO. This immigrants scheme transfers knowledge from old environments and, thus, may be advantageous when changes are small to

medium. Furthermore, it may be suitable in slowly changing environments since it needs sufficient time to locate a good optimum which can be useful to the new environment since the global optimum may be similar.

The mutation of the best ant is carried out using the inversion operator, where two cities are randomly selected and the sub-tour between them is reversed. However, there are two types of inversion: 1) the simple one is as explained above; and 2) the adaptive one is based on the inver-over operator [10]. The adaptive inversion is much more efficient than the simple one, since inversions are carried out under some criteria. This type of inversion has adaptive characteristics which may be more suitable for DOPs.

3.3 Hybrid Immigrants ACO

The hybrid immigrant ACO (HIACO) algorithm uses an immigrants scheme that combines both random and elitism-based immigrants. The replacing policy is the same as in RIACO and EIACO algorithms. However, half of the immigrants are randomly generated and the other half are generated by mutating the best ant. HIACO attempts to combine the merits of both RIACO and EIACO, where one is good on slowly and slightly changing environments and the other on fast and significantly changing environments. Therefore, HIACO may possibly be suitable under all environmental conditions.

4 Simulation Experiments

4.1 Experimental Setup

In the experiments, we compare RIACO, EIACO, and HIACO with P-ACO with its best population update policy, that is, *Age*. All the algorithms have been applied on the `kroA200` problem instance, obtained from TSPLIB[3], which consists of 200 cities. The dynamic environment was generated by taking away half of its cities and constructing a "spare pool" of cities before running the algorithms. Every f iterations, a percentage of m cities were randomly chosen from the spare pool and exchanged with a percentage of m random ones from the actual instance (the other half cities). This way, the size C of the problem instance remains the same through the whole run.

The parameters f and m indicate the frequency and magnitude of dynamic changes, respectively. The f parameter is defined as the number of iterations between two environmental changes. The m parameter is defined as the percentage of selected cities from the spare pool that replaces other cities from the actual instance. The common parameters used for the algorithms were set according to the guidelines in [3, pp. 71] as follows: $\alpha = 1$ and $\beta = 2$ for Eq. (1), and $\tau_{init} = 1/(C-1)$. For P-ACO, K was set to 3 and τ_{max} was set to 1.0 as in [6, 7]. For all three proposed algorithms, K was set to 25, in which we replace 6 ants with immigrants ($\approx 25\%$ of K). Moreover, μ was set to 25 ants for all

[3] Available on http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Table 1. In the first section, values in bold indicate the best results of the overall offline performance. In the second section, "$s-$" or "$s+$" means that the first algorithm is significant better or significantly worse than the second algorithm, respectively, whereas "$\sim$" indicates no significant difference between algorithms

Algorithms & Instances	`kroA200`			
$f = 20, m \Rightarrow$	10%	25%	50%	75%
P-ACO	27339.89	28497.20	29072.95	29290.05
RIACO	25798.46	26016.24	26029.56	25975.49
EIACO	25822.68	26001.00	26018.47	25996.12
HIACO	**25752.20**	**25985.19**	**25961.28**	**25907.79**
$f = 100, m \Rightarrow$	10%	25%	50%	75%
P-ACO	**24284.40**	25010.38	25359.90	25394.80
RIACO	24513.54	24799.98	24903.43	24852.92
EIACO	24455.14	24688.14	**24749.48**	**24682.73**
HIACO	24421.26	**24604.94**	24784.14	24683.38
	t-Test Results			
$f = 20, m \Rightarrow$	10%	25%	50%	75%
P-ACO⇔RIACO	$s+$	$s+$	$s+$	$s+$
P-ACO⇔EIACO	$s+$	$s+$	$s+$	$s+$
P-ACO⇔HIACO	$s+$	$s+$	$s+$	$s+$
RIACO⇔EIACO	$\sim$	$\sim$	$\sim$	$\sim$
RIACO⇔HIACO	$\sim$	$\sim$	$s+$	$s+$
EIACO⇔HIACO	$s+$	$\sim$	$s+$	$s+$
$f = 100, m \Rightarrow$	10%	25%	50%	75%
P-ACO⇔RIACO	$s-$	$s+$	$s+$	$s+$
P-ACO⇔EIACO	$s-$	$s+$	$s+$	$s+$
P-ACO⇔HIACO	$s-$	$s+$	$s+$	$s+$
RIACO⇔EIACO	$\sim$	$\sim$	$s+$	$s+$
RIACO⇔HIACO	$\sim$	$\sim$	$s+$	$s+$
EIACO⇔HIACO	$\sim$	$\sim$	$\sim$	$\sim$

algorithms in order to have the same number of evaluations in each iteration, that is, 25 evaluations per iteration.

For each algorithm on a DTSP instance, $N = 30$ independent runs were executed on the same random environmental changes. The algorithms were executed for $G = 1000$ iterations and the overall offline performance is calculated as follows:

$$\boldsymbol{P}_{offline} = \frac{1}{G}\sum_{i=1}^{G}(\frac{1}{N}\sum_{j=1}^{N}P_{ij}^{*}) \tag{5}$$

where P_{ij}^{*} defines the best ant after a change of iteration i of run j [12]. Our implementation closely follows the guidelines of the ACOTSP[4] framework.

The value of f was set to 20 and 100, indicating environmental changes of high and low frequencies, respectively. The percentage of m was set to 10,

[4] Available on http://www.aco-metaheuristic.org/aco-code

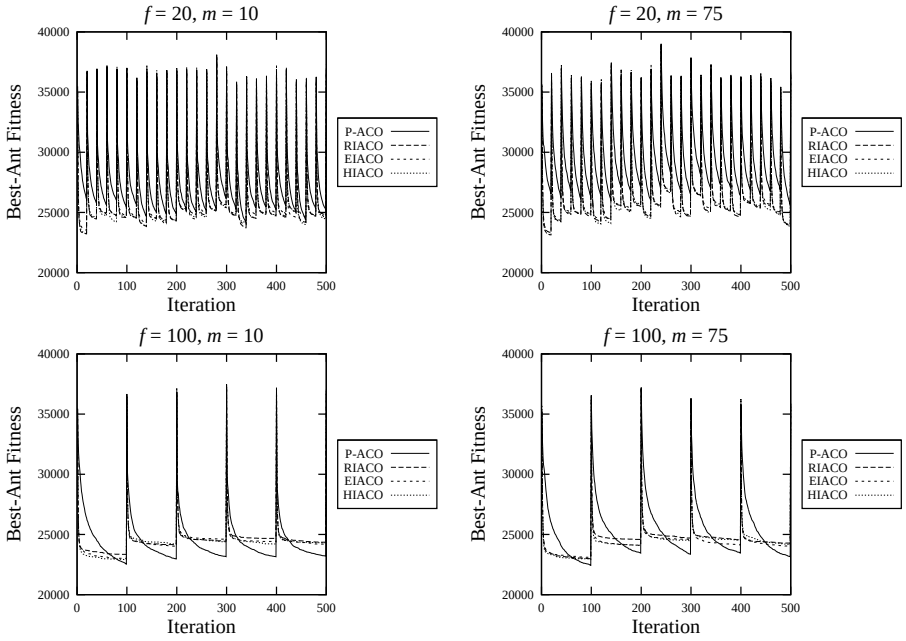

Fig. 1. Overall offline performance for different dynamic test problems.

25, 50, and 75, indicating the degree of environmental changes from small, to medium, to large, respectively. As a result, eight dynamic environments, i.e., 2 values of $f \times 4$ values of m, were generated from the stationary TSP instance to systematically analyze the adaption and searching capability of each algorithm on the DTSP.

4.2 Experimental Results

The experimental results regarding the offline performance of the algorithms with the corresponding statistical results of two-tailed t-test with 58 degrees of freedom at a 0.05 level of significance are presented in Table 1. Moreover, to better understand the dynamic behaviour of the algorithm, the results are plotted in Fig. 1 for the first 500 iterations with $f = 20$, $m = 10$ and $m = 75$, and $f = 100$, $m = 10$ and $m = 75$. From the experimental results, several observations can be made by comparing the behaviour of the algorithms.

First, RIACO, EIACO, and HIACO significantly outperform the P-ACO algorithm in almost all test cases. On cases where the frequency is short the P-ACO algorithm is not able to maintain a population list of useful solutions because it has slow convergence. This can be observed from Fig. 1, where under large frequencies P-ACO converges slowly to a better optimum than other algorithms. However, when the magnitude of changes is small with a large frequency, it is significant better than the other algorithms. On the other hand, RIACO, EIACO

and HIACO are able to provide a good solution faster after a change since they gain more diversity by incorporating immigrants to the population.

Second, RIACO performs slightly better than EIACO on cases where the frequency is small, as expected. This is because EIACO needs to converge to a good optimum in order to be effective. This task needs sufficient time as with the P-ACO algorithm. Recall that in EIACO we use an adaptive inversion, which provides more exploration than the simple inversion. On the other hand, EIACO performs significant better than RIACO in almost all slowly changing environments since it has sufficient time to locate a good solution.

Third, HIACO improves the performance of EIACO and RIACO on cases where the frequency is small. Incorporating random and elitism-based immigrants, diversity is achieved with random ones and the guidance on promising areas in the search space is achieved by the elitism-based ones. As a result, diversity is controlled more since RIACO may generate high levels of diversity and become ineffective due to the lose of useful solutions found during past iterations. However, HIACO is not improving on cases where the change frequency is large, but it keeps the merits of EIACO since they are not significant different.

5 Conclusions

Different types of immigrants schemes have been successfully applied to EAs to address DOPs efficiently. In this paper, we apply random, elitism-based, and hybrid immigrants schemes into ACO for the DTSP, resulting in the RIACO, EIACO, and HIACO algorithm, respectively. The difference of these algorithms lies in the way immigrant ants are generated. The immigrant ants are generated randomly for RIACO and are generated by mutating the best ant of the previous iteration for EIACO, respectively. For HIACO, half of the immigrant ants are generated randomly and the other half are generated using the elitism-based scheme. All immigrants replace the worst ants of the population on every iteration in order to gain sufficient diversity within the population, which can be useful for the DTSP.

Comparing with P-ACO, an existing ACO framework developed for DOPs, on different cases of dynamic environments, the following concluding remarks can be drawn. First, immigrants schemes are advantageous for ACO algorithms. Second, the performance of EIACO is significant better than RIACO in slowly changing environments. Third, the performance of RIACO is slightly better than EIACO on most fast changing environments, while the performance of HIACO is significant better than both of them. Forth, the performance of HIACO on slowly changing environments is competitive with EIACO. Finally, P-ACO may be a sufficient choice in very slowly and slightly changing environments, or in cyclic environments since it is a memory-based approach [16].

For further work, it would be interesting to compare the algorithms on other dynamic environmental cases, i.e., cyclic environments where past environments reappear, and investigate the effect of other parameters or strategies within the proposed algorithms, i.e., which ants should immigrants replace.

Acknowledgement

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E060722/1.

References

1. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York (1999)
2. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. on Syst., Man and Cybern., Part B: Cybern. 26(1), 29–41 (1996)
3. Dorigo, M., Stützle, T.: Ant Colony Optimization. The MIT Press, London, England (2004)
4. Eyckelhof, C. J., Snoek, M.: Ant Systems for a Dynamic TSP. In: ANTS'02: Proc. of the 3rd Int. Workshop on Ant Algorithms, pp. 88–99 (2002)
5. Grefenestette, J. J.: Genetic algorithms for changing environments. In: Proc. of the 2nd Int. Conf. on Parallel Problem Solving from Nature, pp. 137–144 (1992)
6. Guntsch, M., Middendorf, M.: Applying population based ACO to dynamic optimization problems. In: Proc. of the 3rd Int. Workshop on Ant Algorithms, LNCS 2463, pp. 111–122 (2002)
7. Guntsch, M., Middendorf, M.: A population based approach for ACO. In: EvoWorkshops 2002: Appl. of Evol. Comput., pp. 72–81 (2002)
8. Guntsch, M., Middendorf, M.: Pheromone modification strategies for ant algorithms applied to dynamic TSP. In: EvoWorkshops 2001: Appl. of Evol. Comput., pp. 213-222 (2001)
9. Guntsch, M., Middendorf, M., Schmeck, H.: An ant colony optimization approach to dynamic TSP. In: Proc. of the 2001 Gen. and Evol. Comput. Conf., pp. 860–867 (2001)
10. Guo, T., Michalewicz, Z.: Inver-over operator for the TSP. In: Proc. of the 5th Int. Conf. on Parallel Problem Solving from Nature, pp. 803–812 (1998)
11. Holland, J.: Adaption in Natural and Artificial Systems. University of Michigan Press (1975)
12. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments - a survey. IEEE Trans. on Evol. Comput. 9(3), 303–317 (2005)
13. Stützle, T., Hoos, H.: The MAX-MIN ant system and local search for the traveling salesman problem. In: Proc. of the 1997 IEEE Int. Conf. on Evol. Comput., pp. 309–314 (1997)
14. Yang, S.: Genetic algorithms with memory and elitism based immigrants in dynamic environments. Evol. Comput. 16(3), 385–416 (2008)
15. Yang, S.: Genetic algorithms with elitism based immigrants for changing optimization problems. In: EvoWorkshops 2007: Appl. of Evol. Comput., LNCS 4448, pp. 627–636 (2007)
16. Yang, S.: Memory-based immigrants for genetic algorithms in dynamic environments. In: Proc. of the 2005 Genetic and Evol. Conf., vol. 2, pp. 1115–1122 (2005)
17. Yu, X., Tang, K., Chen, T., Yao, X.: Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization. Memetic Comput. 1(1), 3–24 (2009)