Memory-based CHC Algorithms for the Dynamic Traveling Salesman Problem

Anabela Simões[*]
Coimbra Polytechnic
Rua Pedro Nunes - Quinta da Nora
3030-199 Coimbra, Portugal
abs@isec.pt

Ernesto Costa
CISUC, University of Coimbra
Polo II - Universidade de Coimbra
3030-290 Coimbra - Portugal
ernesto@dei.uc.pt

ABSTRACT

The CHC algorithm uses an elitist selection method that, combined with an incest prevention mechanism and a method to diverge the population whenever it converges, allows the maintenance of the population diversity. This algorithm was successfully used in the past for static optimization problems. The use of memory in Evolutionary Algorithms has been proved to be advantageous when dealing with dynamic optimization problems. In this paper we investigate the use of three different explicit memory strategies included in the CHC algorithm. These strategies - direct, immigrant and associative - combined with the CHC algorithm are used to solve different instances of the dynamic Traveling Salesman Problem in cyclic, noisy and random environments. The experimental results, statistically validated, show that the memory schemes significantly improve the performance of the original CHC algorithm for all types of studied environments. Moreover, when compared with the equivalent memory-based standard EAs with the same memory schemes, the memory-based CHC algorithms obtain superior results when the environmental changes are slower.

Categories and Subject Descriptors

I. [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE—*Problem Solving, Control Methods, and Search*

General Terms

Algorithms, Experimentation, Performance

Keywords

Evolutionary Algorithms, Dynamic Environments, Memory

1. INTRODUCTION

In time-varying optimization problems the fitness function, the design parameters or the environmental conditions may change over time. Evolutionary Algorithms (EA) have been successfully used to solve different dynamic optimization problems (DOPs) [3]. The EAs used to cope with DOPs are usually improved with mechanisms that prevent the premature convergence of the population. These improvements include methods to promote the diversity when a change is detected [5], methods to maintain the diversity through the entire run [21], the incorporation of memory [20], the use of multi-populations [3] or the anticipation of the change [12]. The use of memory to enhance EAs for dynamic environments has been proved to be advantageous for many types of DOPs. Memory works by storing useful information from the current environment and reusing it later when a new environment appears. The store of the useful information can be done implicitly, by using redundant representations [14], or explicitly, by storing useful information from the current environment [20].

The CHC algorithm (Cross-generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation) proposed by Eshelman [6] uses an elitism selection method combined with a highly disruptive crossover promoting the diversity of the population. The CHC algorithm was tested against different genetic algorithm approaches, in several static optimization problems, achieving superior results, especially on hard problems [16]. The main characteristic of this algorithm is its capacity of preventing the convergence of the population, a key issue when dealing with dynamic environments. Recently, different CHC algorithms were successfully used for dynamic environments. Those algorithms combined the characteristics of the CHC with immigrant-based techniques and obtained superior results for the studied DOPs [13]. These results indicate that the inherent properties of this algorithm are suitable to dynamic optimization problems and should be further explored. In this paper we propose three improved CHC algorithms for dealing with dynamic environments. The improvements consist of the incorporation of different schemes of explicit memory and a restart mechanism applied whenever a change is detected. The memory schemes store either good solutions and/or environmental information of the current environment using it later to create new individuals suited for the new environment. The introduced enhancements to the CHC algorithm create successful methods for dealing with dynamic environments. The new memory-based CHC algorithms are tested in different instances of the Dynamic Traveling Salesman Problem (DTSP). The traveling salesman problem (TSP) is a well-known NP-hard combinatorial optimization problem, used as benchmark. The DTSP is obtained by deleting

[*]also belongs to CISUC

or inserting some cities or by changing the costs between cities [7], [9]. The experimental results show that the proposed CHC algorithms efficiently solve different instances of the DTSP and the statistical comparison of the results with other EAs variants validates the efficiency of the proposed algorithms.
The rest of the paper is organized as follows: next section briefly reviews relevant work on the DTSP. Section 3, describes the memory schemes, the implemented algorithms and the peer algorithms used in the experimental study. Section 4 details the experimental setup used in this work. The experimental results and analysis are presented in section 5. Section 6 concludes the paper and some considerations are made about future work.

2. THE DYNAMIC TRAVELING SALESMAN PROBLEM

In the Traveling Salesman Problem (TSP), given a set of cities and their pair wise distances, the goal is to find the shortest possible tour that visits each city exactly once. The Dynamic Traveling Salesman Problem (DTSP) is a generalization of the classic TSP where changes can be introduced by adding or deleting new cities, swapping the location of the cities or changing the values of the pair wise distances. When a change is introduced, the salesman has to re-plan his route. The objective is to minimize the expected total cost, i.e. the sum of the distances used to visit the entire tour. Since the introduction of the DTSP by Psaraftis in [10], several evolutionary approaches have been proposed to solve this problem. Guntsch and Middendorf [7] introduced a population-based ant colony optimization algorithm to solve the DTSP and investigated three strategies for pheromone modification. Younes et al. [23] presented a benchmark generator for DTSP and several EAs were compared under different instances of the DTSP. Zhou et al. [24] proposed three different operators that, using previous information about the current environment, enhanced the performance of EAs for DTSPs. Li et al [4] presented an improved inver-over operator based on a gene pool, which stores a set of most promising gene segments by applying heuristic rules. Yan et al. [17] proposed a new algorithm based on the inver-over operator for TSP and used this method to successfully solve different instances of the DTSP. An immune system-based GA called PISGA was investigated in [9]. The proposed method combined a permutation-based dualism scheme in the clone process and a memory-based vaccination approach to further improve its performance for DTSP. Recently, Wang et al. [15] presented an agent-based evolutionary search algorithm for solving DTSP. In the proposed method all the agents of the current population co-evolve to track the dynamic optima. In [13] different CHC algorithms empowered with immigrant-based techniques were used in different instances of the DTSP.

3. IMPLEMENTED ALGORITHMS

In this paper we analyze and test three memory schemes combined with the CHC algorithm. The original CHC algorithm is described in the next section.

3.1 The CHC algorithm

The original binary-coded CHC was proposed by Eshelman [6] and its main idea is the combination of an elitism selection strategy with a highly disruptive crossover, promoting a high diversity into the population. The algorithm works with a population of individuals and, at every step, a new set of solutions is produced by selecting pairs of solutions from the population (the parents) and recombining them. The mating pool is created by giving to each individual in the population the chance to reproduce. So, the parent population is formed with all the individuals of the current population, but in a random order. The CHC algorithm uses an incest prevention mechanism: the parent population is paired for crossover but, before mating, the Hamming distance between the potential parents is calculated and if half this distance does not exceed a difference threshold d, they are not mated and no offspring is created. The CHC doesn't use mutation, but only a highly disruptive recombination mechanism called Half Uniform Crossover (HUX) that combines exactly half of the non-matching alleles, where the bits to be exchanged are chosen at random. This method guarantees that the two offspring are always at the maximum Hamming distance from their two parents, resulting in the introduction of a high diversity in the new population avoiding the risk of premature convergence. The next population is built using an elitist selection mechanism: p members of the current population (p is the population size) are merged with the generated offspring and the best p individuals are selected to compose the new population. When a parent and an offspring have the same fitness value, the former is preferred to the latter. The difference threshold d is usually initialized to $L/4$ (L is the chromosome length). If no offspring is obtained in one generation, d is decremented by one, indicating that the population is converging. When the difference threshold d drops to zero, a restart process, substituting the usual mutation operator is executed. This step consists of the re-initialization of the population: the best individual is preserved and the remaining individuals are created by randomly changing a percentage (defined by the divergence rate dr) of the best individual's alleles.

```
Function $CHC$
    $L$: chromosome length
    $p$: population size
    $dr$: divergence rate
    $d$: difference threshold

$t = 0$; $d = L/4$; $Initialize(P(0))$
repeat
    $Evaluate(P(t))$
    Preserve best individual from $P(t-1)$
    $P'(t) = Selection\_CHC(P(t))$
    $C(t) = Crossover\_CHC(P'(t))$
    $Evaluate(C(t))$
    $newP(t) = Select_{best}(P(t), C(t))$
    if  $newP(t) = P(t)$
        decrement $d$
    if $d < 0$
        $newP(t) = Reinitialize(newP(t), dr)$
        $d = L/4$
    $P(t) = newP(t)$
    $t = t + 1$
until stop_condition
```

Figure 1: Pseudo code for the CHC algorithm

Eshelman extended the CHC algorithm to permutation representations: this algorithm, similar to the previously described, uses a crossover operator that creates a single child by preserving the edges that the parents have in common and then randomly assigns the remaining edges in order to generate a legal solution. Incest prevention is made by allowing crossover only when the number of common edges of the two parents is greater than the difference threshold d. This algorithm is also included in our empirical study and will be denoted as CHC. Fig. 1 presents the pseudo code for the CHC algorithm.
This paper improves Eshemaln's original CHC algorithm through the incorporation of different memory mechanisms and an addition step that reinitializes the population whenever a change is detected. This reinitialization is the same process that is performed when the parameter d is equal to zero. Everything else is used as in the original CHC algorithm. The next section details the implemented memory schemes: direct, associative and immigrant-based.

3.2 Direct Memory Algorithms

A **direct memory scheme** is characterized by storing good solutions of the current environment and reusing them when an environmental change is detected [20]. This type of memory scheme has been widely used in EAs for dynamic environments, e.g., [2] or [22]. The direct memory scheme is used as follows: the population and the memory are initialized at random. The memory is updated when a change happens, storing the best individual from the population just before the change. To update the memory, the random individuals, generated at the beginning are randomly selected and replaced first. If no random individual is found, the elite from the previous population replaces the closest memory point, if it is a better solution (according to the previous environment). This closest solution is found using the *similar* replacing strategy [3] which selects the most similar individual in terms of Hamming distance. When a change in the environment occurs, a new set of individuals is formed by merging the memory and the search population. Then, these individuals are evaluated in the context of the new environment, and the best p (population size) individuals are selected to become the new search population. Through this process, the memory remains unchanged. At every generation, the best individual from the previous population is preserved and transferred to the next population replacing the worst individual (elitism of size 1). This type of memory is used in the CHC algorithm described before, which is referred as Direct Memory CHC algorithm (DMCHC) and in a standard Evolutionary Algorithm, which will be called Direct Memory Evolutionary Algorithm (DMEA) and is similar to the algorithms proposed by Branke [2] and Yang [20]. In the DMCHC algorithm, when a change is detected, the population is reinitialized using the divergence rate dr, as explained in the previous section. After that, the reinitialized population is merged with the memory and the best individuals are selected as the new population. Moreover DMEA and DMCHC differ in the evolutionary process: the first uses the traditional tournament selection, order crossover and swap mutation and the second uses the CHC's evolutionary process described formerly. The pseudo code of DMEA and DMCHC is presented in Fig. 2.

```
Function $DMCHC$ and $DMEA$
    $L$: chromosome length
    $p$: population size; $m$: memory size
    $n$: global number of individuals
    $dr$: divergence rate
    $d$: difference threshold

$t = 0$; $d = L/4$;
$Initialize(P(0))$
$Initialize(M(0))$
repeat
    $Evaluate(P(t))$
    $Evaluate(M(t))$
    Preserve best individual from $P(t-1)$
    if change is detected then
        if DMCHC then
            $P(t) = Reinitialize(P(t), dr)$
        $P'(t) = RetrieveBest(P(t), M(t))$
        $M(t) = UpdateMemory(M(t), P(t-1))$
    if DMCHC then
        $P'(t) = Selection\_CHC(P(t))$
        $C(t) = Crossover\_CHC(P'(t))$
        $Evaluate(C(t))$
        $newP(t) = Select_{best}(P(t), C(t))$
        if $newP(t) = P(t)$ then
            decrement $d$
        if $d < 0$
            $newP(t) = Reinitialize(newP(t), dr)$
            $d = L/4$
    if DMEA then
        $P'(t) = Tournament\_Selection(P(t))$
        $C(t) = Order\_Crossover(P'(t))$
        $newP(t) = Mutation(C(t))$
    $P(t) = newP(t)$
    $t = t + 1$
until stop_condition
```

Figure 2: Pseudo code for DMCHC and DMEA

3.3 Immigrant Memory Algorithms

The **immigrant memory scheme** was proposed by [18] and explicitly stores the best individual of the current environment, using it to create immigrants that are introduced into the population at every generation. Therefore, when a change in the environment occurs, no information from the memory is retrieved, but it is expected that the diversity created by the immigrants helps the EA readapting to the new environment. The immigrant memory scheme is used as follows: the population and the memory are initialized at random. The memory is updated in the same way as described for the direct memory scheme. The memory retrieval is not dependent on the detection of environmental changes. The memory is reevaluated every generation and the best individual from the memory is used to create immigrants that are introduced into the main population. The number of immigrants is a percentage (r_i) of the total number of individuals (n). The immigrants are created by mutating the best memory individual using an established mutation probability (p_i). Those immigrants are introduced into the population replacing the worst $r_i \times n$ ones. This immigrant memory is used in the CHC algorithm, which is referred as Immigrant Memory CHC algorithm (IMCHC), and in a standard EA, which will be called Immigrant Memory Evolutionary Algorithm (IMEA) and is almost identical to the algorithm proposed by Yang [18]. As before, in the IMCHC algorithm, when a change is detected, the population is reinitialized using the divergence rate dr. The differences in the evolutionary process, stated for DMEA and DMCHC, are also true for IMEA and IMCHC. The pseudo code of IMEA and IMCHC is presented in Fig. 3.

```
Function $IMCHC$ and $IMEA$
    $L$: chromosome length
    $p$: population size; $m$: memory size
    $n$: global number of individuals
    $r_i$: ratio of immigrants
    $p_i$: probability for immigrants creation
    $dr$: divergence rate
    $d$: difference threshold

$t = 0$; $d = L/4$;
$Initialize(P(0))$
$Initialize(M(0))$
repeat
    $Evaluate(P(t))$
    $Evaluate(M(t))$
    Preserve best individual from $P(t-1)$
    if change is detected then
        if IMCHC then
                $P(t) = Reinitialize(P(t), dr)$
        $M(t) = UpdateMemory(M(t), P(t-1))$
    $P_i(t) = createImmigrants(P(t), M(t), r_i \times n, p_i)$
    $Evaluate(P_i(t))$
    $P(t) = ReplaceWorst(P(t), P_i(t), r_i \times n)$
    if IMCHC then
        $P'(t) = Selection\_CHC(P(t))$
        $C(t) = Crossover\_CHC(P'(t))$
        $Evaluate(C(t))$
        $newP(t) = Select_{best}(P(t), C(t))$
        if $newP(t) = P(t)$ then
            decrement $d$
        if $d < 0$
            $newP(t) = Reinitialize(newP(t), dr)$
            $d = L/4$
    if IMEA then
        $P'(t) = Tournament\_Selection(P(t))$
        $C(t) = Order\_Crossover(P'(t))$
        $newP(t) = Mutation(C(t))$
    $P(t) = newP(t)$
    $t = t + 1$
until stop_condition
```

Figure 3: Pseudo code for IMCHC and IMEA

3.4 Associative Memory Algorithms

The **associative memory scheme** was proposed by Yang [19] and Karaman [8] and was inspired in the Population-Based Incremental Learning (PBIL) algorithm proposed by Baluja [1]. In this type of memory scheme the current best individual of the population is stored in the memory as well as the environmental information given by a vector that describes the allele distribution of the population. For instance, in binary representation, this vector gives the frequency of ones over the population at each gene locus. Each memory point consists of a pair $< S, V >$ where S is the stored individual and V is the associated allele distribution vector. The memory is evaluated every generation and if a change is detected, the best individual in the memory $< B_m, V_m >$ is extracted and used to create a set of $\alpha \times n$ new individuals using the vector V_m. Those new individuals are inserted into the population replacing the worst ones. The parameter $\alpha \in [0, 1]$ is called associative factor and determines the number of individuals created from the memory when a change happens. The memory updating mechanism is analogous to the one used in direct and immigrant memory schemes. Every time the memory is updated, the pair $< B_p, V_p >$ is created and stored in the memory. B_p is the current best individual of the population and V_p is the allele distribution vector of the actual population. This pair replaces a random point in the memory if one still exists, or the similar point, otherwise. For a representation based on permutations, such is the case of DTSP, the allele distribution vector stores, for each position, the most common allele in the population for that position. If an allele is already stored in a position, it cannot be used in another position. When an environmental change is detected this vector is used to create new individuals as follows: the permutation stored in the vector is used as a first individual and the remaining are created through swap mutation of this individual using a probability p_m. The CHC algorithm using the associative memory will be called as Associative Memory CHC algorithm (AMCHC). The standard EA using the same memory scheme will be referred as Associative Memory Evolutionary Algorithm (AMEA) and is analogous to the associate memory algorithm used in [20]. As before, in the AMCHC algorithm, when a change is detected, the population is reinitialized using the divergence rate dr. The differences in the evolutionary process, stated for DMEA, DMCHC, IMEA and IMCHC, are also true for AMEA and AMCHC. The pseudo code of AMEA and AMCHC is presented in Fig. 4.

4. EXPERIMENTAL DESIGN

4.1 Dynamic TSP

Experiments were carried out on different DTSPs. In this study, we adopted the method proposed in [23] to create DTSP instances based on the data of kroA100 [11]. Younes' generator uses three modes for creating DTSPs: insert/delete mode (IDM), city swap mode (CSM) and edge change mode (ECM). This paper used ECM mode to create different DTSPs by changing the values of the pair wise distances. This type of change reflects a real world problem called traffic jam, where the distances between cities are viewed as the time needed to travel between them. In ECM, an initial instance of the TSP is chosen to start the run and the changes consist of modifying the costs of a set of edges, using a user defined factor a. The edges, which costs are to be increased, must be selected from the best individual and the edges to decrease cannot belong to the best individual. The change period (r) is measured by the number of function evaluations between changes and the severity of the change (s) is controlled by the number of edges that are changed. In order to test cyclic change periods, the increase/decrease of the edge costs is applied alternatively: during a cycle of length l the costs of the edges are increased by a certain amount a. After that, the costs of the edges are decreased during a cycle of the same length. The decrease phase consists of the removal of the previously introduced changes in reverse order, i.e., after a cycle of length l, the increments of the costs are removed so the instances return to the previous states. These cyclic change periods are different

```
Function AMCHC and AMEA
    L: chromosome length
    p: population size
    m: memory size
    n: global number of individuals
    α: associative factor
    dr: divergence rate
    d: difference threshold

t = 0; d = L/4;
Initialize(P(0))
Initialize(M(0))
repeat
    Evaluate(P(t))
    Evaluate(M(t))
    Preserve best individual from P(t-1)
    if change is detected then
        if AMCHC then
            P(t) = Reinitialize(P(t), dr)
        I(t) = CreateIndividuals(M(t), α × n)
        P(t) = ReplaceWorst(P(t), I(t), α × n)
        V(t) = CreateDistributionVector(P(t-1))
        M(t) = UpdateMemory(M(t), V(t), P(t-1))
    if AMCHC then
        P'(t) = Selection_CHC(P(t))
        C(t) = Crossover_CHC(P'(t))
        Evaluate(C(t))
        newP(t) = Select_best(P(t), C(t))
        if newP(t) = P(t) then
            decrement d
        if d < 0
            newP(t) = Reinitialize(newP(t), dr)
            d = L/4
    if AMEA then
        P'(t) = Tournament_Selection(P(t))
        C(t) = Order_Crossover(P'(t))
        newP(t) = Mutation(C(t))
    P(t) = newP(t)
    t = t + 1
until stop_condition
```

Figure 4: Pseudo code for AMCHC and AMEA

from Younes' original generator, where there is only a cycle where the costs are increased and a second phase where the changes are removed in reverse order. We also study noisy and random environments: in noisy environments, the increase/decrease phases are applied as in cyclic environments, but the matrix of the distances is slightly changed using a noisy factor. In random environments, at every change step, the edges to increase or to decrease are decided at random using a uniform distribution.

4.2 Parameters Setting

The parameters of the algorithms are set using typical values found in similar studies: for DMEA, IMEA and AMEA, a generational replacement with elitism of size one is used, combined with the tournament selection with size two. Crossover is applied with probability $p_c = 70\%$ and swap mutation (for DMEA, IMEA and AMEA) with a probability $p_m = 1\%$. In order to have the same number of function evaluations per generation, the global number of individuals n is set as follows: for DMCHC, DMEA, AMCHC and AMEA, $n = 100$. For IMCHC and IMEA, $r_i \times n$ immigrants are also evaluated every generation, so n is computed using the expression $n = \frac{100}{1+r_i}$. The memory size is $m = 20\% \times n$. The ratio of immigrants r_i used in IMCHC and IMEA is $r_i = 20\%$ and the immigrants are created using swap mutation with a rate set to $p_i = 1\%$. For the associative memory schemes, the value of α is set to 0.5. Several instances of DTSP are tested: different change periods of size $r \in \{1000, 5000, 10000\}$ function evaluations and, for each case, different severities of the change $s \in \{10\%, 20\%, 50\%, 80\%, 100\%\}$. The number of edges that are changed defines the severity of the change. A constant amount $a = 25\%$ is used to increase (or decrease) the distances between two cities i and j as follows: $dist_{i,j} = dist_{i,j} + dist_{i,j} \times a$. For cyclic and noisy environments, the cycle length to increase/decrease the distances is set to $l = 5 \times r$. The noisy factor is set to 1%. The divergence rate used in DMCHC, IMCHC and AMCHC is set to $dr = 20\%$. A change in the environment is detected when a modification in the matrix of distances is observed. For each experiment of an algorithm, 30 runs are executed for 200 environmental changes. The overall performance used to compare the algorithms is the offline performance [3] averaged over 30 independent runs. The statistical validation was made using the nonparametric Friedman test at a 0.01 level of significance. After this test, the multiple pair wised comparisons were performed using the Nemenyi procedure with Bonferroni correction.

5. RESULTS

Fig. 5, Fig. 6 and Fig. 7 show the results obtained for different instances of cyclic, noisy and random DTSPs, respectively. The corresponding statistical results of comparing algorithms are given in Table 1 for cyclic DTSP, in Table 2 for cyclic DTSP and in Table 3 for random DTSP. The notation used in these tables is $+$, $-$ or $\sim$, when the first algorithm is significantly better than, significantly worse than, or statistically equivalent to the second algorithm, respectively. The statistical results refer only to the comparison of the proposed methods among them and with the peer algorithms. No comparisons between CHC, DMEA, IMEA and AMEA are presented. The obtained results show that all memory approaches significantly improve the original CHC algorithm, for all types of environments. Fig. 5, Fig. 6, Table 1 and Table 2 show that, for cyclic and noisy environments, the results are almost equivalent. For those cases, in rapidly changing environments ($r = 1000$), the use of the CHC algorithm combined with the different memory schemes presents worse results when compared with their peer algorithms DMEA, IMEA and AMEA. For $r = 1000$ the best results are achieved by DMEA. For larger change periods, DMCHC and AMCHC obtain the best results, while the worst results are attained by DMEA and IMEA. IMCHC outperforms DMEA and IMEA, but obtains worse results than AMEA. For random environments, the results presented on Fig. 7 and Table 3 show that, once again, for $r = 1000$, the memory-based CHC algorithms obtain worse results than the remaining algorithms. The best results, for rapidly changing random environments are achieved by DMEA. When the changes in the environment become slower, the investigated CHC algorithms present superior performances: DMCHC and AMCHC obtain the best results and IMCHC obtains better performances than DMEA and IMEA. The worst performances are achieved by DMEA and IMEA. In order to better un-

Table 1: The statistical results on cyclic DTSP

	$r = 1000$					$r = 5000$					$r = 10000$				
severity ⇒	0.1	0.2	0.5	0.8	1.0	0.1	0.2	0.5	0.8	1.0	0.1	0.2	0.5	0.8	1.0
DMCHC - CHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
DMCHC - DMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
DMCHC - IMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
DMCHC - AMEA	−	−	−	−	−	+	+	∼	∼	+	+	+	+	+	+
DMCHC - IMCHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
DMCHC - AMCHC	−	−	−	−	−	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
IMCHC - CHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
IMCHC - DMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
IMCHC - IMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
IMCHC - AMEA	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
IMCHC - AMCHC	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
AMCHC - CHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
AMCHC - DMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
AMCHC - IMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
AMCHC - AMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+

Table 2: The statistical results on noisy DTSP

	$r = 1000$					$r = 5000$					$r = 10000$				
severity ⇒	0.1	0.2	0.5	0.8	1.0	0.1	0.2	0.5	0.8	1.0	0.1	0.2	0.5	0.8	1.0
DMCHC - CHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
DMCHC - DMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
DMCHC - IMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
DMCHC - AMEA	−	−	−	−	−	+	+	∼	+	+	+	+	+	+	+
DMCHC - IMCHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
DMCHC - AMCHC	−	−	−	−	−	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
IMCHC - CHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
IMCHC - DMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
IMCHC - IMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
IMCHC - AMEA	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
IMCHC - AMCHC	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
AMCHC - CHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
AMCHC - DMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
AMCHC - IMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
AMCHC - AMEA	−	−	−	−	−	+	+	∼	+	+	+	+	+	+	+

derstand these results, the diversity of the population was measured. Table 4 presents the diversity for the different types of environments using severity equal to $s = 100\%$ (the diversity of the population obtained with other severities was similar). Observing the diversity of the population, there is not a direct correlation between diversity and performance. The presented results indicate that the proposed memory-based CHC algorithms need extra time to evolve and find good solutions. If the environmental changes are too fast, the algorithm has not enough time to correctly explore the search space. The high diversity of the population obtained by the memory-based CHC algorithms when $r = 1000$ combined with the inferior results indicate that the evolutionary process is being continuously disrupted. When the time between changes is larger, the diversity promoted by the memory-based CHC algorithms decreases and the evolutionary process is able to find better solutions. Nevertheless, the promotion of the diversity is not the only factor influencing the results. The used memory schemes are also controlling the obtained results. In fact, for $r = 5000$ and $r = 10000$, IMEA promotes higher diversity than AMEA, but its performance is worse. Moreover, for random environments, in rapidly changing environments, DMEA preserves the lower diversity, but obtains the best results.

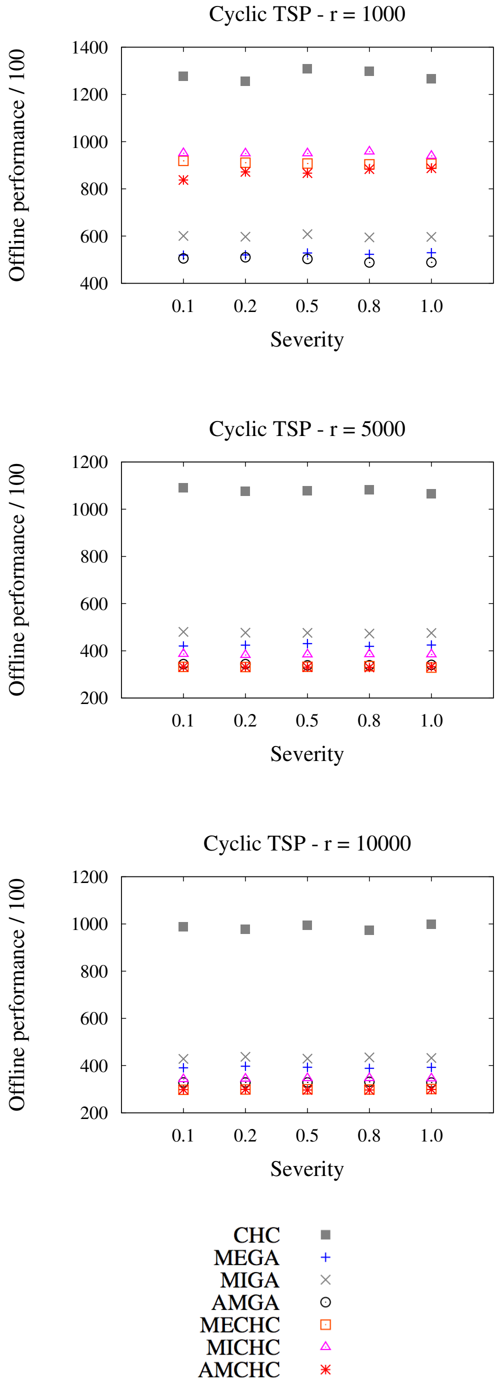

Figure 5: Offline performance for the cyclic DTSP

6. CONCLUSIONS AND FUTURE WORK

This paper investigates the use of three different memory schemes in the CHC algorithm. Direct, immigrant-based and associative memory approaches are combined with the CHC algorithm aiming to solve different instances of the DTSP. The memory-based CHC algorithms were compared with standard evolutionary algorithms using the same memory strategies and also with Eshelman's original CHC algorithm. The empirical study used cyclic, noisy and random instances of the DTSP, and different change periods and change severities were analyzed. The experimental results show that the improved CHC algorithms outperformed the original CHC algorithm for all the studied situations. Moreover, the results demonstrate that the memory-based CHC algorithms are not suitable for rapidly changing environments, but outperform the standard EAs for slowing changing environments. The direct and the associative memory schemes used in the CHC algorithm obtained better results

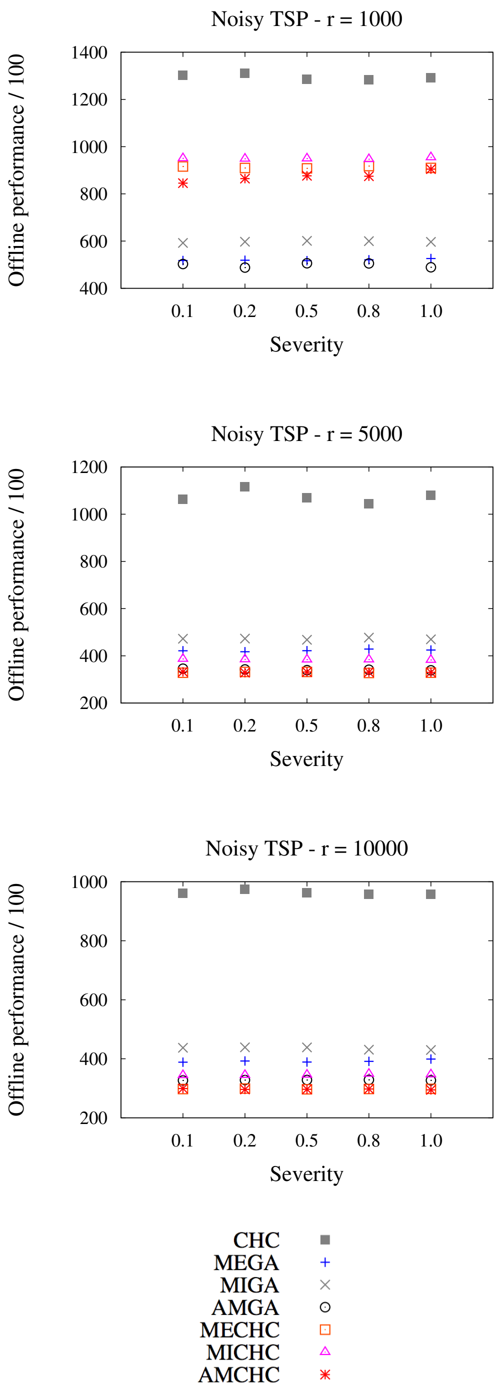

Figure 6: Offline performance for the noisy DTSP

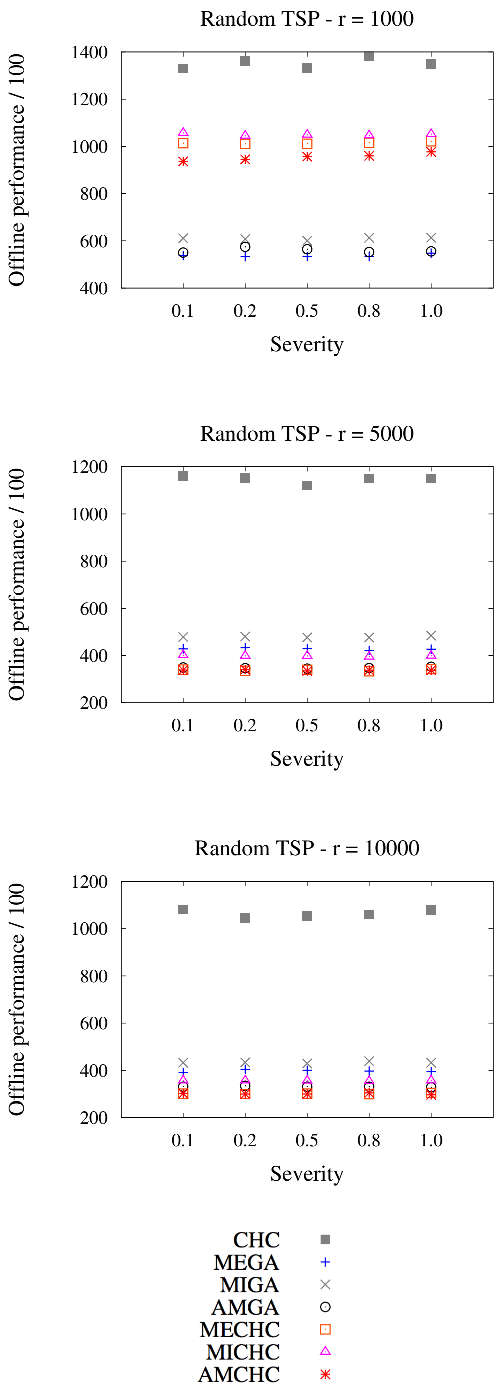

Figure 7: Offline performance for the random DTSP

than the immigrant-based scheme. In general, the associative memory scheme achieved the best results, while the immigrant memory method attained the worst scores. As future work we intend to analyze the sensitivity of the divergence rate on the performance of CHC algorithms and to use these algorithms in other problems using different representations.

7. REFERENCES

[1] S. Baluja. Population-Based Incremental Learning: a method for integrating genetic search based function optimization and competitive learning. Technical Report *TR CMU-CS-94-163*, Carnegie Mellon University, 1994.

[2] J. Branke. Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems. In

Table 3: The statistical results on random DTSP

	$r = 1000$					$r = 5000$					$r = 10000$				
severity ⇒	0.1	0.2	0.5	0.8	1.0	0.1	0.2	0.5	0.8	1.0	0.1	0.2	0.5	0.8	1.0
DMCHC - CHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
DMCHC - DMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
DMCHC - IMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
DMCHC - AMEA	−	−	−	−	−	∼	∼	∼	∼	∼	+	+	+	+	+
DMCHC - IMCHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
DMCHC - AMCHC	−	−	−	−	−	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
IMCHC - CHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
IMCHC - DMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
IMCHC - IMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
IMCHC - AMEA	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
IMCHC - AMCHC	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
AMCHC - CHC	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
AMCHC - DMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
AMCHC - IMEA	−	−	−	−	−	+	+	+	+	+	+	+	+	+	+
AMCHC - AMEA	−	−	−	−	−	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼

Table 4: Diversity of the population

	r	CHC	DMEA	IMEA	AMEA	DMCHC	IMCHC	AMCHC
Cyclic	$r = 1000$	0.23	0.18	0.43	0.78	0.91	0.65	0.77
	$r = 5000$	0.22	0.04	0.38	0.26	0.72	0.65	0.70
	$r = 10000$	0.22	0.02	0.38	0.13	0.65	0.57	0.64
Noisy	$r = 1000$	0.23	0.18	0.43	0.78	0.91	0.64	0.79
	$r = 5000$	0.22	0.04	0.38	0.26	0.72	0.65	0.70
	$r = 10000$	0.22	0.02	0.38	0.13	0.65	0.57	0.64
Random	$r = 1000$	0.24	0.25	0.47	0.80	0.91	0.72	0.77
	$r = 5000$	0.23	0.05	0.39	0.27	0.71	0.65	0.69
	$r = 10000$	0.22	0.03	0.38	0.13	0.61	0.56	0.61

Proceedings of the IEEE Congress on Evolutionary Computation, pages 1875–1882. IEEE Press, 1999.

[3] J. Branke, T. KauBler, C. Schmidt. A Multi-Population Approach to Dynamic Optimization Problems. In I. Parmee (ed) , *Proceedings of Adaptive Computing in Design and Manufacture*, pages 299–308. Springer-Verlag, 2000.

[4] C. Li, M. Yang, L. Kang. A New Approach to Solving Dynamic Traveling Salesman Problems. In Wang, T.-D. and et al. (eds), *Proceedings of 6th International Conference on Simulated Evolution and Learning*, pages 236–243, LNCS 4247. Springer, 2006.

[5] H. G. Cobb. An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms having Continuous, Time-Dependent Nonstationary Environments. *TR AIC-90-001*, Naval Research Laboratory, 1990.

[6] L. J. Eshelman. The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In Gregory J. E. Rawlins (ed), *Foundations of Genetic Algorithms I*, pages 265–283. Morgan Kaufmann, 1991.

[7] M. Guntsch, M. Middendorf. A Population Based Approach for ACO, In S. Cagnoni and et al. (eds), editors, *Applications of Evolutionary Computing*, LNCS 2279, pages 72–81. Springer-Verlag, 2002.

[8] A. Karaman, S. Uyar, G. Eryigit. The Memory Indexing Evolutionary Algorithm for Dynamic Environments. In *Applications of Evolutionary Computing*, LNCS 3449, pages 563–573. Springer-Verlag, 2005.

[9] L. Liu, D. Wang, S. Yang. An Immune System Based Genetic Algorithm Using Permutation-Based Dualism for Dynamic Traveling Salesman Problems. In M. Giacobini and et al. (eds), *Applications of Evolutionary Computing*, LNCS 5484, pages 725–734. Springer-Verlag, 2009.

[10] H. N. Psaraftis. Dynamic Vehicle Routing Problems. In B. L. Golden and A. A. Assad (eds), *Vehicle Routing: Methods and Studies*, pages 223–248. Elsevier, 1988.

[11] G. Reinelt. TSPLIB. University of Heidelberg. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`, 1995.

[12] A. Simões, E. Costa. Prediction in Evolutionary Algorithms for Dynamic Environments using Markov Chains and Nonlinear Regression. In *Proc. of the 11th Int. Genetic and Evolutionary Computation Conference*, pages 883–890. ACM Press, 2009.

[13] A. Simões, E. Costa. CHC-based Algorithms for the Dynamic Traveling Salesman Problem. In C. Di Chio et al. (eds.), EvoApplications 2011, Part I, LNCS 6624, pages 354–363. Springer-Verlag, 2011.

[14] A. S. Uyar, A. E. Harmanci. A New Population Based Adaptive Dominance Change Mechanism for Diploid Genetic Algorithms in Dynamic Environments. *Soft Computing*, 9(11): 803–814, Springer, 2005.

[15] D. Wang, S. Liu. An Agent-based Evolutionary Search for Dynamic Traveling Salesman Problem. In *International Conference on Information Engineering*, pages 111–114. IEEE Press, 2010.

[16] D. Whitley, S. Rana, J. Dzubera, E. Mathias. Evaluating Evolutionary Algorithms. *Artificial Intelligence*, Number 85, pages 245–276, 1996.

[17] X-S. Yan, H-M. Liu, J. Yan, Q-H. Wu. A Fast Evolutionary Algorithm for Traveling Salesman Problem. In *Proceedings of the 3rd International Conference on Natural Computation*, pages 85–90. IEEE Press, 2007.

[18] S. Yang. Memory-based Immigrants for Genetic Algorithms in Dynamic Environments. In Hans-Georg Beyer (ed). *Proceedings of the Seventh International Genetic and Evolutionary Computation Conference*, pages 1115–1122. ACM Press, 2005.

[19] S. Yang. Population-based Incremental Learning with Memory Scheme for Changing Environments. In Hans-Georg Beyer (ed). *Proceedings of the Seventh International Genetic and Evolutionary Computation Conference*, pages 711–718. ACM Press, 2005.

[20] S. Yang. Explicit Memory Schemes for Evolutionary Algorithms in Dynamic Environments. In S. Yang et al. (eds). *Evolutionary Computation in Dynamic and Uncertain Environments*, pages 3–28. Springer-Verlag, 2007.

[21] S. Yang. GAs with Elitism-based Immigrants for Changing Optimization Problems. In M. Giacobini et al. (eds). *Applications of Evolutionary Computing*, LNCS 4448, pages 627–636. Springer-Verlag, 2007.

[22] S. Yang. Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. *Evolutionary Computation*, 3(16): 385–416, MIT Press, 2008.

[23] A. Younes, O. Basir, P. Calamai. A Benchmark Generator for Dynamic Optimization. In *Proceedings of the 3rd Int. Conf. on Soft Computing, Optimization, Simulation & Manufacturing Systems*, 2003.

[24] A. Zhou, L. Kang, Z. Yan. Solving DTSP with Evolutionary Approach in Real Time. *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, pages 951–957. IEEE, 2003.