On the Effects of Incorporating Memory in GC-AIS for the Set Cover Problem

Joshi, Ayush; Rowe, Jonathan E.; Zarges, Christine

Published in:
MIC 2015: The XI Metaheuristics International Conference

Publication date:
2015

Citation for published version (APA):
Joshi, A., Rowe, J. E., & Zarges, C. (2015). On the Effects of Incorporating Memory in GC-AIS for the Set Cover Problem. In *MIC 2015: The XI Metaheuristics International Conference* University of Lille 1.

On the Effects of Incorporating Memory in GC-AIS for the Set Cover Problem

Ayush Joshi, Jonathan E. Rowe, Christine Zarges

School of Computer Science
University of Birmingham, Edgbaston, Birmingham, B152TT, UK
{axj006, J.E.Rowe, c.zarges}@cs.bham.ac.uk

Abstract

Learning is an important part of the immune system by which the immune system maintains a memory of the infections it has encountered to protect against future attacks. In this paper we incorporate the mechanism of maintaining a memory in the recently proposed GC-AIS algorithm. GC-AIS has shown good performance on the static set cover problem (SCP) in recent work [13] and we are interested in investigating the merits of GC-AIS in a dynamic setting. We compare the affect of GC-AIS with and without a memory approach on the dynamic SCP instances, which are created with varying degrees of modifications to instances from [2]. Three types of modifications are proposed in the paper by adding, removing or editing the subsets from the original problem instances. It is shown that for the case of adding subsets to the original instance using our memory approach is always beneficial while for the case of removing subsets using our memory approach almost always results in worse performance than when not utilising memory. Finally in the cases with editing subsets it is shown that for lower levels of modification using our memory approach gives better results while when the level of modification is higher our memory based approach is worse than using no memory.

1 Introduction

Many real-world problems have characteristics that change over time in the form of changing objective functions, the problem instance itself or the constraints [25]. Evolutionary dynamic optimisation (EDO) deals with solving these time dependent problems called dynamic optimisation problems (DOP) using evolutionary computation methods. Several techniques have been developed for EDO to tackle these problems like diversity approaches [20], memory [24] and prediction-based [18] approaches, multi-populations [11] and self-adaptive approaches [22]. If the DOP under consideration involves multiple objectives then they are referred to as dynamic multi-objective optimisation problems (DMOP). EDO is a rapidly growing area of research and a lot of work has been done in this field in recent years, a detailed survey of EDO techniques can be found in [25].

Artificial immune systems (AIS) are randomised search heuristics developed from taking inspiration from the immune system of vertebrates. The immune system stands out from other biological systems due to the presence of several desirable properties combined together. It is due to these properties like memory, anomaly detection and robustness that AIS have been applied to very different applications like machine learning, robotics and optimisation. De Castro and Timmis [7] provide a detailed survey of applications. Some work in the field of dynamic optimisation using AIS can be found in [12, 10, 17, 21].

During its lifetime an organism may encounter an infection multiple times. To overcome these infections in the future the immune system maintains a repertoire of cells called memory cells. When an infection occurs for the first time, the immune cells which prove to be the most potent are selected to form memory cells and these are used to fight against subsequent attacks of the same infection. The number of immune cells in the body is regulated and the immune cell repertoire reflects the infections faced by the organism over its lifetime [7]. Memory cells are long lived and provide a long lasting protection against an encountered pathogen. While on one hand multiple exposure to the same pathogen can lead to the production of better antibodies and therefore faster and efficient future immune response, on the other hand it is also known that in some cases the antibodies produced in response to a particular strain of pathogen suppress the creation of new different antibodies in response to a different variant of the pathogen making a person more susceptible to these mutated variant [8]. Exact mechanisms of how memory in the immune system works is a topic of ongoing research.

The Germinal center artificial immune system (GC-AIS) is a novel AIS introduced by Joshi et al. [13], which is inspired by recent research on the germinal centre reaction [26]. The motivation to incorporate memory into GC-AIS comes from the fact that immune memory forms the basis of future immune responses. Using the memory metaphor from the immune system we extend GC-AIS for dynamic environments. Since the effect of memory is not always positive we are interested to study the memory approach to find cases when including memory is useful and when it is not. This new variant of GC-AIS is tested on the SCP with a simple dynamic component, as previous work by Joshi et al. [13] has shown some advantages of using GC-AIS for static SCP and we are interested to study the behaviour of GC-AIS using an additional memory approach in a dynamic setting. The dynamic component is created by altering the instances of the SCP taken from the OR-library [2], by adding, removing or editing subsets from each instance.

2 Preliminaries

This section details the formal definition of the set cover problem and the dynamic extension of the problem. This is followed by a brief overview of memory based approaches for EDO and finally the GC-AIS extended with memory is presented.

As stated in [13], the immune system tries to solve the set cover problem in an abstract way. This acts as the motivation to study the performance of memory based approaches for the GC-AIS on the SCP.

2.1 Set Cover Problem

The set cover problem can be defined as: Given a *universe set* U, which consists of m items, and another set S, which contains n subsets of U and whose union equals U, the problem is to find the smallest subset of S, which covers U. A more formal definition is as follows:

Definition 1 *Let the set of* m *items* $U := \{u_1, ..., u_m\}$ *denote the universe and let* $S := \{s_1, ..., s_n\}$ *such that* $s_i \subseteq U$ *for* $1 \leq i \leq n$ *and* $\bigcup_{i=1}^{n} s_i = U$. *The uni-cost set cover problem can be defined as finding a selection* $I \subseteq \{1, 2, ..., n\}$ *such that* $\bigcup_{k \in I} s_k = U$ *with minimum* $|I|$.

SCP is a constrained single objective problem where the objective is to find the smallest subset of S that covers U with the constraint that the subset covers U. It is a NP hard combinatorial optimisation problem, which has many practical applications, an important one being scheduling [4]. Caprara et al. [4] provide a survey of techniques employed to solve the set cover problem.

A multi-objective formulation of the set cover problem can be more efficient in finding the optimal solution than a single objective formulation [15] as it makes the algorithm behave like Chvatal's greedy algorithm [6]. A multi-objective formulation can be obtained by transforming the constraint as a secondary objective [9]. Let $X = x_1 x_2 \dots x_n$, with $x_i \in \{0, 1\}$ for $1 \leq i \leq n$, denote a solution to the problem where $x_i = 1$ if set s_i is in the solution and 0 otherwise. Let N be the number of subsets selected in X and C be the number of elements left uncovered in U. The fitness function for this multi-objective formulation of SCP can now be defined as $F = \langle C, N \rangle$.

If we visualise a possible pathogen as an instance of the universe set, and the binding of the B cells as possible solutions, then the immune system tries to solve the problem of finding the best match to the pathogen, by randomised variations in the solutions. Joshi et al. [13] have investigated the performance of GC-AIS on the SCP using its multi-objective formulation and compared it with the parallel global simple evolutionary multi-objective optimiser.

2.1.1 A Simple Dynamic Extension for SCP

Another definition of SCP based on matrices can be stated as follows.

Definition 2 ([1]) *Let $A = (a_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$ with $a_{ij} \in \{0, 1\}$ be a matrix with m rows and n columns where a column j is said to cover a row i if $a_{ij} = 1$. Let $X = x_1 \dots x_n \in \{0, 1\}^n$ denote a solution where $x_j = 1$, $j = 1, \dots, n$, if column j is in the solution and 0 otherwise. Minimise $\sum_{j=1}^{n} x_j$ subject to $\sum_{j=1}^{n} a_{ij} x_j \geq$ for all $i = 1, \dots, m$.*

The rows in this definition correspond to the universe set U and the columns correspond to S, the subsets of U from Definition 1. Definition 2 is introduced as the modifications for the dynamic model in this paper are based on this matrix definition and several existing dynamic SCP models use Definition 2, e. g. [5]. In order to convert the static SCP to a dynamic problem some form of time dependent modification must be introduced. By extending Definition 2 a formal definition of dynamic SCP based on work by Chrissis et al. [5] can be stated as:

Definition 3 ([5]) *Let $X_t = x_{1t} \dots x_{nt}$ denote a binary string, which represents a solution at time t, and n_t the number of columns in the matrix $A_t = (a_{ijt})_{1 \leq i \leq m, 1 \leq j \leq n_t}$ at time t. Minimise $\sum_{t=0}^{T} \sum_{j=1}^{n_t} x_{jt}$ subject to $\sum_{j=1}^{n} a_{ijt} x_{jt} \geq 1$ for all $i = 1, \dots, m$.*

Three different types of modifications are introduced in this paper namely adding, removing or editing columns in the $m \times n$ matrix according to Definition 2 which correspond to changes in the matrix a_{ijt} at different times in Definition 3. According to Definition 1 these changes can be seen as adding elements to S, removing elements from S or editing elements inside S. For each type of modification suggested the level of change is varied from low to high and for each such level of change 30 instances are created. In this paper these new instances are called novel instances and they represent the changed problem. The algorithm is first run on the original instance which represents the time $t = 0$ and at time $t = 1$ a novel instance is presented to the algorithm. This can be seen as a dynamic SCP formed by a sequence of static instances joined together and two time states namely $t = 0$ and $t = 1$. At $t = 0$ the original instance is presented to GC-AIS and the best obtained solution is recorded. At time $t = 1$ a novel instance is presented to the two algorithms, the original GC-AIS and GC-AIS with memory, and the performance of the two is compared.

Some work using dynamic SCP has been done by Chrissis et al. [5] for dynamic facility locations and Kodani et al. [19] for real time fault diagnosis.

2.2 EDO and Memory-Based Approaches

DOPs are characterised by time-dependent changes to the problem, which are usually in the form of a sequence of static problems linked by some dynamic rules or having a time dependent parameter in its expression [25]. The key difference between DOPs and static problems is the requirement of evolutionary algorithms (EAs) to track changing optima for DOPs rather than simply locating them.

To solve the issues of tracking optima several techniques have been developed in the EDO literature, memory being one of them. Memory schemes utilise storing good solutions and re-using them at a later stage. They are useful when changes are recurrent or periodical and old optima may be revisited, see, e. g., [24].

2.2.1 Memory-Based Techniques for EDO

These approaches are often used when the changes in the DOPs are recurrent or periodic in nature, therefore an old optimum may be revisited in the future [16, 24]. In such cases it makes sense to save old solutions as a form of memory and use them when an old optima is encountered again in order to save computation time. Two main variants of memory approaches exist in literature namely: implicit and explicit memory [16]. Implicit memory is maintained by encoding the chromosome as multiploid instead of the more common haploid in EAs for static problems. Explicit memory on the other hand involves an external storage of information. This information may be a previously known good solutions where it is called direct memory or associative information where it is called indirect memory. The memory is periodically updated by replacement and based on the current best information. According to [16]

advantages of memory based approaches are their usefulness in periodic environments and maintaining diversity while disadvantages of memory-based approaches are that they are only useful in cases when previous optima are revisited. The redundant coding of memory is not useful in cases when number of fluctuations is high. A review of memory-based approaches can be found in [16]

A small review of memory based approached for DMOP is provided here. Branke [3] used a explicit memory approach to store individuals in a finite memory, which uses a replacement approach when the memory becomes full. Yang [23] proposed an associative scheme in where individuals along with a distribution scheme are stored in the memory. Zhang and Qian [27] introduced the dynamic constrained multi-objective artificial immune system (DCMOAIS), which consists of three modules: a problem detection module based on T-cells, a solution module based on B-cells and a storage module based on memory (M-cells).

3 Extended GC-AIS with Memory (m-GC-AIS)

GC-AIS [13] is a new AIS for multi-objective optimisation based on recent understanding of the germinal centre reaction in the immune system. A germinal centre (GC) is a region in the immune system where a type of immune cells called B cells are presented with the invading pathogen in order to generate antibodies (Abs), which fight the infection.

When the body is attacked by a pathogen the number of *GCs* begin to rise in order to generate *Abs* which are capable of eradicating the pathogen. By continuous proliferation mutation and selection of B cells in the germinal centres their ability to bind with the pathogen increases and this reaction is able to produce *Abs*, which can successfully fight the infection. There is periodic communication between *GCs* by transmitting *Ab*s. Towards the stage when *Abs* produced are capable to fight off the infection the number of *GCs* starts declining. The GC-AIS is based on a new theory of selection in the GC reaction proposed by Zhang et al. [26] according to which there is a competition between the mutating B cells and the *Abs* and cells, which are unable to compete, die by the process of natural cell death (apoptosis). This can even lead to whole *GCs* to disappear if cells within them cannot compete with *Abs* from neighbours.

The GC-AIS is extended by a simple explicit memory component for the dynamic SCP proposed above. This memory is a finite store which stores information about the best solution of the original SCP instance before changes have been applied to it. The size of the memory has been restricted to 1 in this study as GC-AIS starts with 1 individual and for the extended model with memory this initial individual is replaced by the one from memory. The memory contains all the subsets from S which have corresponding 1s in the best known solution. This memory is then used to initialise the GC at the beginning of Algorithm 1. In the following parts of the paper the extended GC-AIS is referred to as (m-GC-AIS).

Algorithm 1 The GC-AIS WITH MEMORY (M-GC-AIS)

Let G^t denote the population of GCs at generation t and g_i^t the i-th GC in G^t.
Create GC pool $G^0 = \{g_1^0\}$ and initialise g_1^0 from memory. Let $t := 0$.
loop
  for each GC g_i^t in pool G^t in parallel **do**
    Create offspring y_i of individual g_i^t by standard bit mutation.
  end for
  Add all y_i to G^t, remove all dominated solutions from G^t and let $G^{t+1} = G^t$.
  Let $t = t + 1$.
end loop
Save best solution information as memory

Based on Algorithm 1 the steps in the m-GC-AIS can be described as follows: A single GC is created at the start, which contains one individual that represents a B-cell. This *GC* is initialised by an external memory component. By standard bit mutation of B-cells in the GC offspring are created,

Problem	$m \times n$	density	Known [14]	Obtained (using GC-AIS)
scp41	200×1000	2%	(0,38)	(0,39)
scp63	200×1000	5%	(0,21)	(0,21)
scpa5	300×3000	2%	(0,38)	(0,39)
scpb4	300×3000	5%	(0,22)	(0,23)
scpd2	400×4000	5%	(0,25)	(0,25)
scpnre1	500×5000	10%	(0,17)	(0,17)

Table 1: Best known and obtained solutions for the original problem instances. Density refers to the percentage of 1s in the $m \times n$ matrix. The column *known* contains solutions obtained by [14] while column *obtained* contains solutions obtained using GC-AIS.

where standard bit mutation refers to each bit being flipped with probability $1/n$. There is a migration of fitness values of offspring between *GC*s at every generation which corresponds to migration of *Ab*s. After this all dominated solutions are deleted, which can be seen as cell death of B cells which cannot compete with neighbours and the surviving offspring form new *GC*s. Thus the model is dynamic in nature as the number of *GC*s can change with time. At every generation of the GC-AIS maintains a set of non-dominating solutions.

4 Experimental Set-up

In this section we describe the experimental set-up used for this study. The m-GC-AIS is compared with the standard GC-AIS on some dynamic SCP instances. The dynamic SCP instances are created by modifying 6 static SCP instances selected from the OR-library [2]. The SCP instances in the OR library are grouped into classes based on the size of the instances and the selected instances each belong to a different class, namely 4, 6, A, B, D and RE.

4.1 Novel Instance and Memory Generation

Since the global optima of the original 6 instances are not known we select good solutions to these instances by running GC-AIS and use these as memory for m-GC-AIS. To be more precise, we run GC-AIS on each of the original 6 instances 30 times each for 20,000 generations. To ensure that the selected solutions are sufficiently good we compare them with results from [14] as a measure of closeness to their known best results. This process can be seen as running both GC-AIS and m-GC-AIS on the dynamic SCP at time step $t = 0$, i. e. on the original problem as in the case of m-GC-AIS at time $t = 0$ there is no previous memory therefore it starts from the all $0s$ bit string just like GC-AIS. The selected solutions obtained by GC-AIS are depicted as *Obtained* in Table 1 along with instance sizes, density and the best know solutions from [14].

The best obtained individual is converted to memory by mapping the $1s$ in the solution bit string to the problem instance and storing the corresponding subsets as memory information. It is not enough to save the solution bit string as memory as the size of the novel instance to solve may change therefore the same subsets may not correspond to the same bit positions in the novel instances.

The novel instances are generated by modifying the original instances from [2] by applying 3 types of changes: adding columns, removing columns and editing columns in the matrix.

- Adding columns: For each of the 6 original instances, novel instances are created by adding columns to the $m \times n$ matrix. In other words subsets are added to S while keeping the density of the novel instance the same as the original instance. 30 novel instances are created each by adding k columns where $k \in \{10, 20, 30, \ldots, 100\}$ subsets to the original instances, making the novel instances of size $m \times (n + k)$.

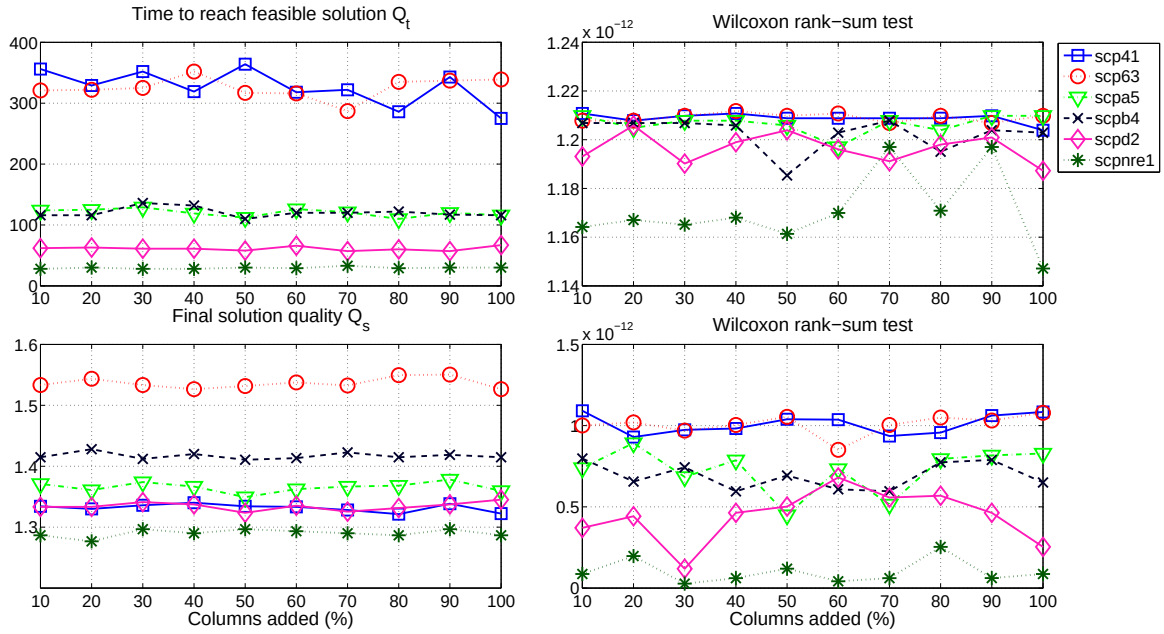

Figure 1: Plots for results obtained on novel instances with added columns. Plots for solution quality shows Q_s and time to reach feasible solution shows Q_t averaged for 30 novel instances for each value of k. Plots for the Wilcoxon rank-sum test show the p value for Q_s and Q_t.

- Removing columns: Removing columns for each of the 6 original instances is performed by utilising the best solution obtained. Only the columns which correspond to a 1 in the solution bit string are removed. This ensures that the optimal solution for the original instance is no longer a feasible solution for the novel instance. 30 novel instances are created each by removing k columns where $k \in \{0.1c, 0.2c, \ldots, 0.9c\}$ and c is the original solution quality, i. e., the number of columns selected (see column *Obtained* in Table 1). Thus reducing the novel instance size to $m \times (n - k)$.

- Editing columns: Editing column is performed by moving individual items from one subset to another. Using only the columns which correspond to a 1 in the solution bit string items are selected randomly from these columns and moved to other randomly selected columns which do not have these items in them. Let d be the total number of $1s$ in the matrix corresponding to the original solution from *Obtained*. 30 novel instances are created each by moving k items where $k \in \{0.1d, 0.2d, \ldots, 0.8d\}$, of the items from the solution columns of the original instances. It should be noted that in this case the size of the novel instances created is the same as the original instances while the optimal solution for the original is no longer feasible in most cases, except when the swaps occur for elements which have duplicates in other columns in the solution.

5 Results and Discussion

In order to compare the performance of m-GC-AIS and GC-AIS in a dynamic setting, both these algorithms are run on the created novel instances which can be seen as running the algorithms on the dynamic SCP at time $t = 1$ where the problem has changed and the m-GC-AIS has memory available from the previous runs of GC-AIS at time $t = 0$. A stopping limit of 1200 generations is set for each algorithm based on observations from the initial runs for time $t = 0$, where it was seen that this number is roughly double the number of generations to reach the feasible solution region. Each algorithm is run for 30 independent runs on each novel instance created and the averages of the sets used and uncovered sets are recorded. The quotient of average solution qualities (Q_s) at the end of the stopping criteria is plotted

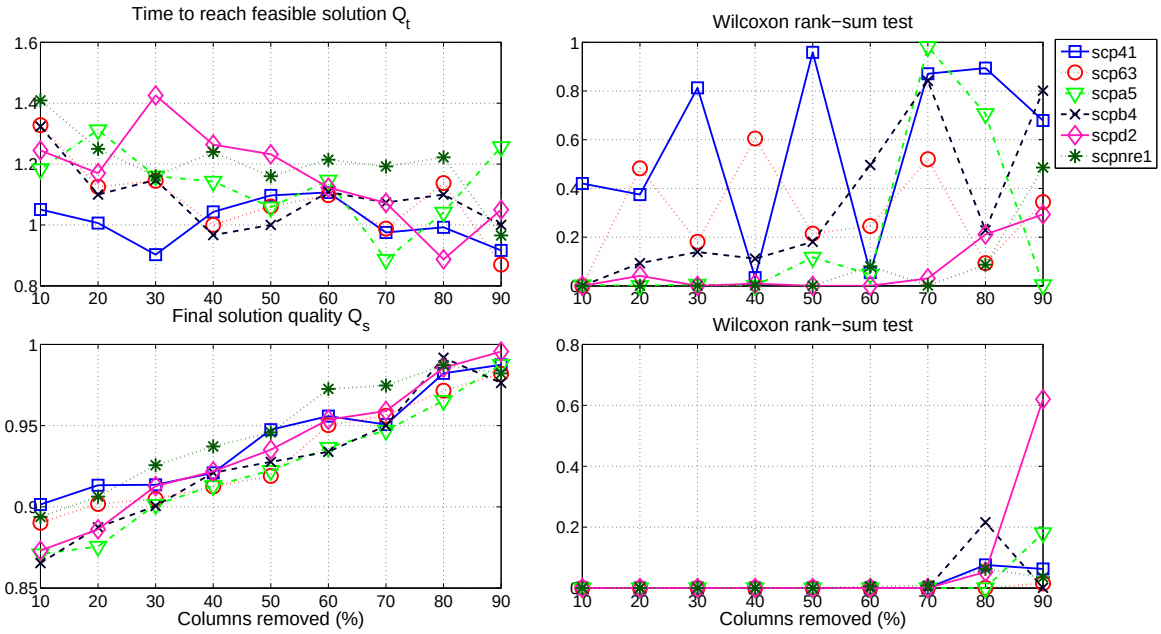

Figure 2: Plots for results obtained on novel instances with columns removed shown as % of c on x-axes. Plots for solution quality shows Q_s and time to reach feasible solution shows Q_t averaged for 30 novel instances for each value of k. Plots for the Wilcoxon rank-sum test show the p value for Q_s and Q_t.

along with the quotient of time taken to reach the feasible region (Q_t). Let $N^{\text{GC-AIS}}$ denote the sets used by GC-AIS and $N^{\text{m-GC-AIS}}$ be the sets used by m-GC-AIS we define Q_s as $N^{\text{GC-AIS}}/N^{\text{m-GC-AIS}}$. Let $T^{\text{GC-AIS}}$ denote the time taken by GC-AIS to reach feasible region and $T^{\text{m-GC-AIS}}$ be the time taken by m-GC-AIS we define Q_t as $T^{\text{GC-AIS}}/T^{\text{m-GC-AIS}}$. The Wilcoxon rank-sum test is used as a measure to test the statistical difference between the two algorithms. The test is performed for Q_s and Q_t for both the algorithms for the 30 novel instances for each value of k in the respective modification. The p values obtained from the test are plotted and a value of 0.05 is used a significance level to state the statistical difference of the algorithms.

5.1 Adding Columns

Adding columns can be seen as the most trivial case out of the three modifications as in some sense even though the problem size has changed at $t = 0$ the obtained best solution still remains a feasible solution. From Figure 1 it can be seen that the plots for Q_s is always greater than 1, meaning that the solution obtained by GC-AIS always has more sets used than solutions obtained by m-GC-AIS. This is not surprising since m-GC-AIS is able to find the possible best solution from the first generation based on the solution from memory. The plots for Q_t are always $\gg 1$ which is clear as the time taken by m-GC-AIS is always 1 since the memory solution is a feasible solution. The p values from the statistical test show that in all the test cases the two algorithms are significantly different. Based on these results it can be said that using memory is clearly the best approach when only the problem size changes but the old solution remains feasible.

5.2 Removing Columns

The plots for Q_s from Figure 2 at the end of the stopping criteria show an increase from values 0.85-0.9 for $k = 0.1c$ to almost 1 for $k = 0.9c$. This means that for smaller values of k the number of sets used by GC-AIS is lower than the number of sets used by m-GC-AIS and as k approaches $0.9c$ Q_s is almost 1. The statistical test show that the two algorithms are significantly different till $k = 0.7c$. This

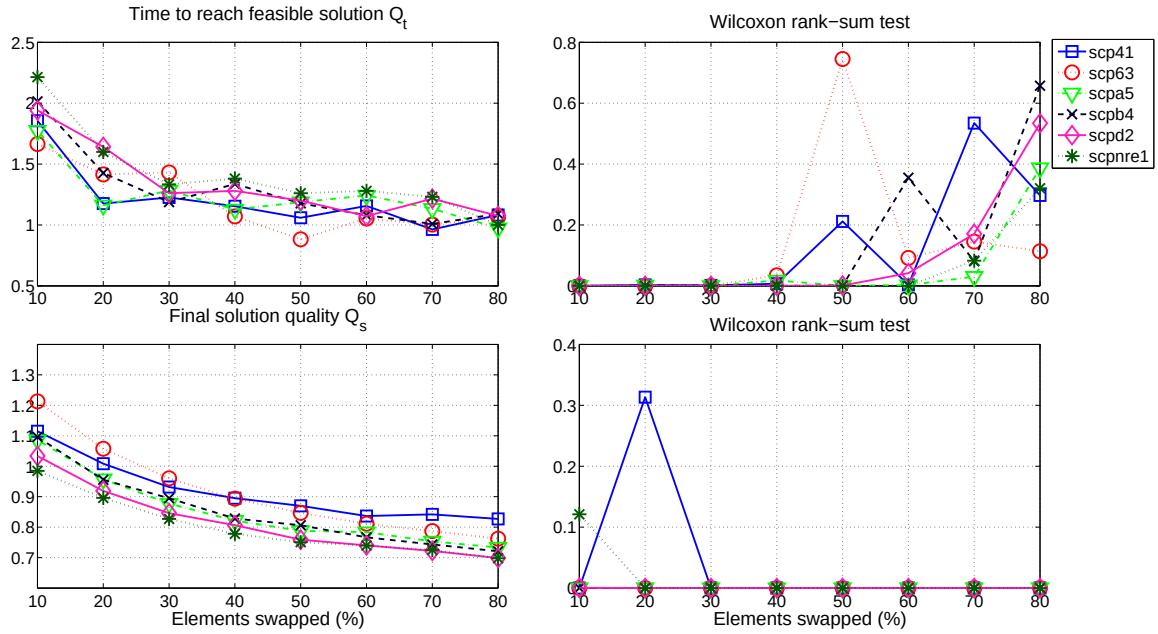

Figure 3: Plots for results obtained on novel instances with elements swapped from subsets shown as $\%$ of d on the x-axes. Plots for solution quality shows Q_s and time to reach feasible solution shows Q_t averaged for 30 novel instances for each value of k. Plots for the Wilcoxon rank-sum test show the p value for Q_s and Q_t.

behaviour can be interpreted as starting from the all 0s bit string in the case of GC-AIS is always better than starting form a memory in m-GC-AIS which in this case behaves like a randomly generated string since it no longer a feasible solution for the novel instance. The plots for Q_t on the other hand do not reveal a pattern and no conclusive statement can be made on these which is confirmed by the p values from the statistical tests.

5.3 Editing

Results for editing columns show a more clear pattern of observable behaviour which can be seen in Figure 3. The plots for Q_s show some interesting results, for values of k up to $0.2d$ the ratios plotted are > 1 meaning that GC-AIS uses more sets than m-GC-AIS therefore using memory is preferred but for $k > 0.2d$ the ratio is < 1 meaning that the solution quality of GC-AIS is better than m-GC-AIS. The plots for the statistical test show that the two algorithm are statistically different when Q_s is not ≈ 1. A possible explanation of this behaviour can be made as for smaller editing the memory solution may require either no addition or very few addition of new sets to make the solution feasible again. No new solutions may be required for the cases when the moved item from one set could be available in another subset hence not needing any changes, while a few changes are needed when a moved item was no longer available in the subsets in the solution and a new subset is required to be added. In the case of larger changes the memory solution almost behaves as a random solution. These results can be interpreted as for low values of k using the memory approach is preferred while for larger values of k it is almost always better to start from the all $0s$ string rather than have a memory. The plots for Q_t shows a slight decrease from values of 2 for $k = 0.1d$ to 1 for $k = 0.8d$. This means that GC-AIS takes more time to find the feasible region than m-GC-AIS initially for lower values of k and as k increases the two algorithms almost take the same amount of time. This is evident from the p values where for only lower values of k up to $0.3d$ to $0.4d$ the algorithms are statistically different.

6 Conclusions

We are interested in finding the usefulness of memory approaches for GC-AIS on the dynamic SCP. Based on three different modifications to static SCP instances dynamic SCP problems are created and the performance of GC-AIS and m-GC-AIS was compared. In the case of adding columns based on Figure 1 it can be seen that using the memory approach is always better than starting from the all $0s$ bit string. This is due to the fact that the way the problem is created the obtained best solution for the original instance remains feasible for the novel instance and the memory is able to exploit this fact from the beginning of the run. When considering removing columns, based on the solution quality at the stopping criterion it can be said that using memory gives poor performance when compared with starting from the all $0s$ bit string. For the instances with editing based on Figure 3 it is shown from the plots for the time taken to reach the feasible region that for little to moderate editing the feasible region is reached faster when using memory while for larger editing both approaches take similar time. Based on the plots for the solution quality it is shown that using memory gives better results when the level of editing is lower while for larger editing using memory should be avoided.

We have shown the cases where using memory is suitable and when the use of memory should be avoided for dynamic SCP, this is interesting to study as the we would like to learn the behaviour of memory as it seems to be problem size dependant. For future work we would like to investigate further into the case with removing columns by incorporating more instances and individually considering instances with similar density. We believe that instances with similar density might behave similarly and differences could potentially be made clear between the instances with different density. At this stage a very simple model of dynamic SCP is considered and we would like to investigate these findings further on different dynamic problems like the dynamic knapsack problem. We would also like to consider other models of memory storage as a future work.

References

[1] J. E. Beasley. An algorithm for set covering problem. *European Journal of Operational Research*, 31(1):85–93, 1987.

[2] J. E Beasley. OR-library: distributing test problems by electronic mail. *Journal of the operational research society*, pages 1069–1072, 1990.

[3] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *CEC*. IEEE, 1999.

[4] A. Caprara, P. Toth, and M. Fischetti. Algorithms for the set covering problem. *Annals of Operations Research*, 98(1-4):353–371, 2000.

[5] J. W Chrissis, R. P Davis, and D. M Miller. The dynamic set covering problem. *Applied Mathematical Modelling*, 6(1):2–6, 1982.

[6] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[7] L. Nunes de Castro and J. Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, 2002.

[8] M. W Deem and H. Y Lee. Sequence space localization in the immune system response to vaccination and disease. *Physical review letters*, 91(6):068101, 2003.

[9] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. Approximating covering problems by randomized search heuristics using multi-objective models. *Evolutionary Computation*, 18(4):617–633, 2010.

[10] A. Gasper and P. Collard. From GAs to artificial immune systems: improving adaptation in time dependent optimization. In *CEC*, volume 3. IEEE, 1999.

[11] C. Goh and K. Chen Tan. A competitive-cooperative coevolutionary paradigm for dynamic multi-objective optimization. *Evolutionary Computation*, 13(1):103–127, 2009.

[12] E. Hart and P. Ross. An immune system approach to scheduling in changing environments. In *GECCO*, pages 1559–1566. Morgan Kaufmann, 1999.

[13] A. Joshi, J. E. Rowe, and C. Zarges. An immune-inspired algorithm for the set cover problem. In *PPSN XIII*, pages 243–251. Springer, 2014.

[14] N. Musliu. Local search algorithm for unicost set covering problem. In *Proc. of Advances in Applied Artificial Intelligence*, pages 302–311. Springer, 2006.

[15] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity*. Natural Computing Series. Springer, 2010.

[16] T. T Nguyen, S. Yang, J. Branke, and X. Yao. Evolutionary dynamic optimization: Methodologies. In *Evolutionary Computation for Dynamic Optimization Problems*, pages 39–64. Springer, 2013.

[17] A. Simões and E. Costa. An immune system-based genetic algorithm to deal with dynamic environments: diversity and memory. In *Artificial Neural Nets and Genetic Algorithms*, pages 168–174. Springer, 2003.

[18] A. Simões and E. Costa. Improving prediction in evolutionary algorithms for dynamic environments. In *GECCO*, pages 875–882. ACM, 2009.

[19] S. Singh, A. Kodali, K. Choi, K. R Pattipati, S. M Namburu, S. C. Sean, D. V Prokhorov, and L. Qiao. Dynamic multiple fault diagnosis: Mathematical formulations and solution techniques. *Systems, Man and Cybernetics, Part A: Systems and Humans*, 39(1):160–176, 2009.

[20] R. Tinós and S. Yang. A self-organizing random immigrants genetic algorithm for dynamic optimization problems. *Genetic Programming and Evolvable Machines*, 8(3):255–286, 2007.

[21] K. Trojanowski and S. T Wierzchoń. Immune-based algorithms for dynamic optimization. *Information Sciences*, 179(10):1495–1515, 2009.

[22] R. K Ursem. Multinational GAs: Multimodal optimization techniques in dynamic environments. In *GECCO*, pages 19–26. ACM, 2000.

[23] S. Yang. Associative memory scheme for genetic algorithms in dynamic environments. In *Applications of evolutionary computing*, pages 788–799. Springer, 2006.

[24] S. Yang, H. Cheng, and F. Wang. Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile ad hoc networks. *Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(1):52–63, 2010.

[25] S. Yang and X. Yao. *Evolutionary Computation for Dynamic Optimization Problems*. Springer, 2013.

[26] Yang Z., M. Meyer-Hermann, L. A. George, M. T. Figge, M. Khan, M. Goodall, S. P. Young, A. Reynolds, F. Falciani, A. Waisman, C. A. Notley, M. R. Ehrenstein, M. Kosco-Vilbois, and K. Toellner. Germinal center B cells govern their own fate via antibody feedback. *The Journal of Experimental Medicine*, 210(3):457–464, 2013.

[27] Z. Zhang and S. Qian. Artificial immune system in dynamic environments solving time-varying non-linear constrained multi-objective problems. *Soft Computing*, 15(7):1333–1349, 2011.