Using Simple Ancestry to Deter Inbreeding for Persistent Genetic Algorithm Search

Aditya Wibowo and Peter Jamieson
Dept. of Electrical and Computer Engineering
Miami University

Abstract—*In this work, we explore how a mechanism for recording ancestry helps avoid inbreeding and, ultimately, convergence for persistent optimization problems. We focus our experimentation on the traveling salesman problem and introduce a tabu search "like" mechanism in a CHC algorithm and preselection genetic algorithm. We then compare how this mechanism improves the diversity within the solution population. We compare this mechanism to a basic genetic algorithm and show how the quality of results is improved and convergence is delayed. Our results indicate that the CHC algorithm with the inbreeding avoidance mechanism is the current best implementation for persistent optimization problems in maintaining diversity of solutions and to find the best solutions. Preselection shows improvement with our mechanism, but does not seem to have sufficient exploitation to find quality results. Our overall goal is to find the best way to maintain diversity while finding good solutions for single-threaded genetic algorithms.*

1. Introduction

There is a small subset of optimization problems that we call persistent optimization problems (POPs), and these problems are characterized by problems that can have their solution space continuously searched for better solutions. One of the most recent of these types of problems is persistent computer aided design (CAD) for Field-Programmable Gate Arrays (FPGAs). FPGAs are programmable chips that can be updated in the field with new and better designs. The placement stage of FPGA CAD, which tries to pack hardware structures that are connected to one another, can be algorithmically solved using genetic algorithms (GAs), and researchers have explored persistently searching for better placement solutions (improving power consumption) using a GA [1], [2]. Other examples of where persistent optimization algorithms may be useful include energy control and distribution, financials, and data mining. In each of these domains, the solution space is dynamically changing over time, and better optimizations for the problems may result in saved money and higher efficiency. The key question for POPs is whether the additional run-time costs justifies the benefit of potential improved solutions and the incremental cost savings.

POPs fit well into GA frameworks since GAs can be manipulated in terms of exploration versus exploitation phases to continually cross a solution space. Still, convergence [3] [4] defined as the lack of diversity in a population such that new offspring are not sufficiently diverse, therefore, resulting in suboptimal solutions, is a major concern for POPs in addition to all GAs. For problems such as the FPGA placement problem [5] and the Traveling Salesman Problem (TSP), where a genome is expressed as unique string of individual genomes, traditional algorithms such as CHC [6], which are built to preserve diversity, are not directly applicable to these problems since the hamming distance measure of familiarity does not apply.

In this work, we implement versions of CHC and preselection GAs to solve the TSP, and we include an inbreeding avoidance technique inspired by Tabu Search [7]. Our goal is to develop a single threaded GA that avoids convergence for the longest period possible while still generating good results. With the improvements of these algorithms, we plan to further investigate divergence techniques such as the island model [8] to build a larger system for POPs using GAs.

Our results show that our inbreeding avoidance mechanism does achieve higher diversity for both preselection GAs and the CHC algorithm based on a greater number of generations before the the problem stabilizes. Preselection algorithms with inbreeding avoidance last the most number of generations, but it seems that this crowding technique loses some of the advantages of competition/exploitation that the CHC algorithm achieves.

The remainder of this paper is organized as follows. Section 2 describes various techniques to maintain diverse populations for GAs and the relevance of crossbreeding operator to our problems. Section 2.2 describes our inbreeding avoidance technique and the two algorithms that they are implemented within. Section 4 describes our experimental setup and Section 5 shows our results. Finally, Section 6 concludes this work.

2. Background

In this section, we examine various approaches to avoiding premature convergence and the crossover operator for ordered chromosomes.

2.1 Approaches to Avoiding Premature Convergence

Premature convergence is a well known problem with GAs [3] [4]. This problem has been addressed using a number of methods including basic approaches such as:

1) Increasing population size

2) Island models/ Niching [8]
3) Crowding [9]
4) Preselection [10]
5) Inbreeding prevention [6], [11]

The first approach, increasing population size, impacts memory usage and algorithmic run-time, but this approach can be used in combination with all the other approaches to the premature convergence problem. The second approach, Island models, divides the search into a number of parallel solutions where each is run independently of one another. In this way, there is no sharing of genetic material of individuals from one island population to another. This approach can also be applied to any other approach for improving diversity. Therefore, the first two approaches can be used to improve our results and will be considered, in a larger system, once we wish to build a larger system to solve POPs, but our focus for this work is single-threaded approaches at fixed population size that maintain genetic diversity while finding good solutions.

The last three schemes in the list are some examples of algorithmic approaches to the premature convergence problem. Crowding works by having offspring compete against individuals in the population that are most similar and thus maintaining niche lineages within the populations. Similar individuals are found by making a comparison of their genomic strings, and this comparison comes at a computation cost. Preselection is a similar approach to crowding, but to avoid the computational cost an assumption is made; the assumption is that a parent will be a similar individual, and therefore parent and child will compete against each other. Finally, the CHC algorithm takes a number of steps towards avoiding premature convergence, and of main interest to this work, parents of similar genome structure are not bred together (inbreeding prevention). These three approaches have been extensively studied for problems that have a binary encoded genome, and this work looks at two of these approaches for genomes that have unique chromosome encoding (described in the next section).

As mentioned before, the CHC algorithm is a non-traditional GA that was created to avoid premature convergence. In this paper, we implement the CHC algorithm [6] as one of the comparison points for our inbreeding avoidance technique. The CHC algorithm has four main components:

1) Parents and children combined together in competition for next population
2) Inbreeding avoidance by comparing binary encoded genomes hamming distance
3) Highly disruptive crossover operator
4) Full restart (from the best individual) once no new offspring are created in a generation

Since our work is focused on genomes that are not a simple binary encoded string, we take a modified approach to this algorithm. In particular, we use a crossover method described in the next sub section and our inbreeding avoidance technique is described in the section following the background.

2.2 Crossover Operators for Permutation Based Genomes

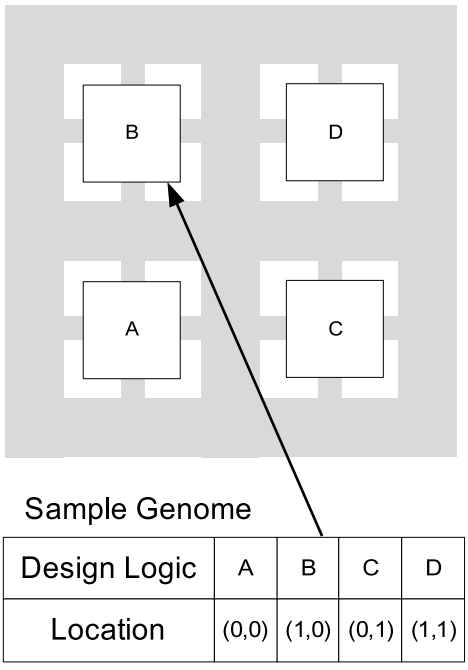

Fig. 1
SAMPLE OF A SMALL PLACEMENT GENOME

In the case of both the TSP and the FPGA placement problem (among other problems) the genome consists of a permutation in which each chromosome is unique. These types of chromosomes are called ordered chromosomes. In the TSP problem, this genomic string represents the order of a tour; for example, for a four city problem we might see the string A, C, B, D which means this solution will go from city A to C, C to B, and B to D in that order. For the placement problem this permutation string indicates which pieces of a circuit are located in a 2D plane. For the previous example, piece A would be placed at x=0, y=0, piece C is at x=0, y=1, piece B is at x=1, y=0, and piece D is at x=1, y=1. Figure 1 shows this example of the 2D placement and the respective genome. This genome structure was originally proposed by Venkatraman *et. al.* [12].

For these types of strings, crossover operators that simply copy the genome of parent 1 and take parts of the genome from parent 2 and map them into the child cannot be used. Instead, careful consideration must be used to perform the crossover. A number of crossover operators of this nature have been proposed and studied ([4], [13], [14], [15], [16], [17], [18], [19]). Cicirello *et. al.* [18] provide a useful classification of these crossover methods by first classifying them as problem dependent or general crossover operators. Cicirello *et. al.* further classify crossovers into three categories (a) position-based crossover (e.g. [16]), (b) order-based crossover (e.g. [19]), and (c) hybrid crossover operators (e.g. [15]).

Recently, the success of a problem dependent crossover operator proposed b Whitley *et. al.* for the TSP suggests that careful thought should be given to a problem with ordered chromosomes. The same is, likely, true for the FPGA placement problem among other problems, and this is an are for future work if we pursue FPGA placement POPs.

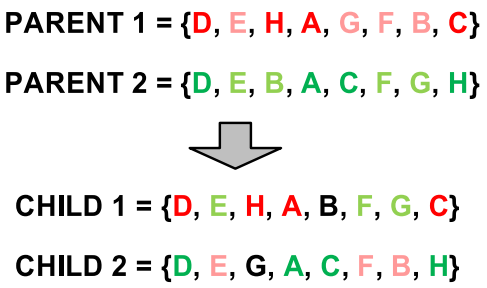

Fig. 2
SAMPLE PMX MUTATION

For this work, we are attempting to find a general single-threaded framework for solving POPs that do not use any of the more modern domain specific solutions that would target the TSP. Instead, we use the partially mapped crossover (PMX) [15], which randomly selects a set of parent genes to be copied to the new child from parent 1. Then, the remaining genes are transferred from parent 2 unless this gene has already been assigned by parent 1. If it has, a reverse mapping using the the information in parent 1 is used to find an appropriate gene to be copied to the empty spot. Figure 2 shows a simple example of the PMX operator where a set of chromosomes have been selected to be copied from parent 1 to child 1 and parent 2 to child 2. The remaining genes are copied over from the opposite parent using a remapping process when needed. Note how, in the figure, child 2 has some chromosomes from parent 1 (illustrated in red), some from parent 2 (illustrated in green), and one remapped chromosome (illustrated in black) . Note that the highly disruptive aspect of the CHC algorithm is not specifically explored in this work, and we simply assume that our crossover operator is sufficient, and we leave this issue as future work, if necessary.

3. Inbreeding Prevention for Permutation Based Genomes

In Eshelman's [6] original work on the CHC algorithm, he introduced inbreeding prevention for binary encoded chromosomes, and in this work we look at a technique to avoid inbreeding for ordered chromosomes and apply this technique both to preselection GAs and our implementation of the CHC algorithm. Our technique is inspired by tabu search [7] where a simple history is kept for previous solutions. In the same way, we can keep ancestry records for each individual by keeping am ancestor tree to a certain depth of generations.

Figure 3 shows how two parents ancestral trees that contain three past generations are combined together in a respective

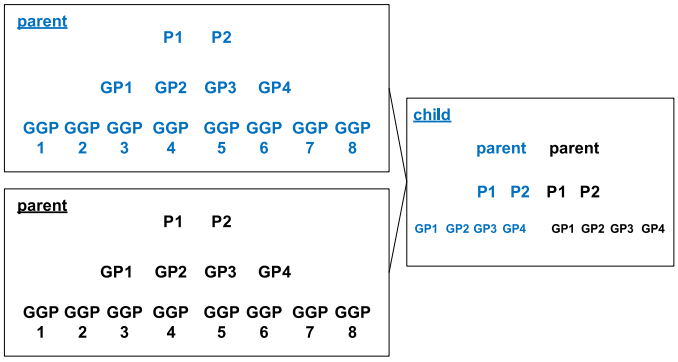

Fig. 3
SHOWS HOW TWO PARENTS ANCESTRY IS RECOMBINED FOR A CHILD

child. Based on our crossbreeding operator, we can assume that both parents contribute roughly 50% of their genetic information to the child. Similarly, grandparents will contribute 25% of their genetic material to the child, and so on for older generations where 4th generation contributes 12.5%, 5th generation contributes 6.25%, and 6th generation contributes 3.125%. Therefore, from a perspective of our inbreeding avoidance it doesn't make much more sense to record deeper than 6 generations, where 6 generations costs us only 128 data locations in memory. This memory cost for ancestry records is small for each member of the population in comparison with the size of their genomic information. Therefore, the memory cost is not significant. We will explore the depth of ancestry in the experimental section.

With the ancestors recorded, avoiding inbreeding is done by comparing two candidate parents and checking if they share any common ancestors. If they do share common ancestors, depending on the GA, a new suitable pair of parents is found or the crossbreeding operation is skipped. In terms of the computation cost to search for shared ancestry, we simply do an exhaustive search of both family trees. We have implemented these trees as arrays, and therefore, the search is very simple. The cost for this comparison is similar to that of calculating the hamming distance between individual genomic strings.

Our inbreeding avoidance mechanism, however, differs from the original approaches in CHC and crowding GAs where the goal is to compare individuals based on how similar they are to one another. Instead, our inbreeding mechanism makes a similar assumption to preselection GAs, where children sharing ancestry will be similar just based on lineage. The problem in implementing a comparison of individuals with ordered chromosomes, like that of the original CHC and crowding GA, is identifying similar solutions, which will lead to implementations of subgraph isomorphism problems [20]. For example, a tour in the TSP might include the substring "A, R, C", and other population solutions with the substring "A, R, C" at some point might be considered similar. Searching for all such string matches would be expensive.

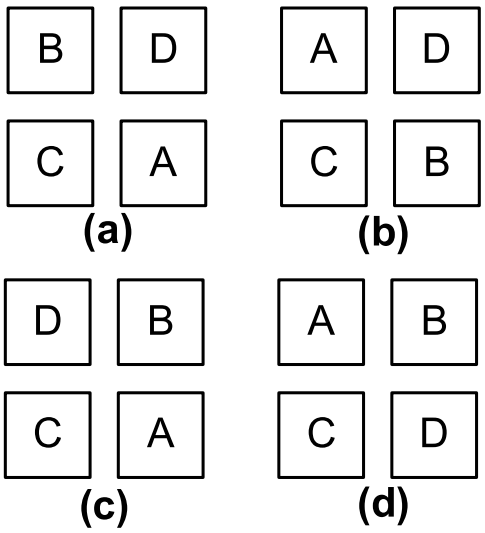

Fig. 4
SHOWS FOUR EXAMPLES OF WHAT MIGHT BE CONSIDERED SIMILAR PLACEMENTS

In the case of the 2D placement problem the problem of finding similar individuals is even harder than finding subgraphs. Figure 4 shows four examples of how 4 things (from a set of, potentially, thousands) can be placed relative close to each other, and in each case we might consider examples (a), (b), (c), and (d) similar to one another since the components are only one hop away from one another. These four components, however, will not be adjacent to one another in the genomic string. For example, example (a) would have a genome of the form "..., B, D, ..., C, A, ..." and (b) would have the form "..., A, D, ..., C, B, ..." meaning sub-string matching approaches cannot be used. Our proposed mechanism, however, can deal with both problems by trading off the measure of similarity for the simplicity of comparing ancestry.

It may also be possible to implement comparison of individuals using some sort of clustering technique, but again, the computational complexity of these approaches is high.

In the original implementation of the CHC algorithm, as the population becomes more and more similar the algorithm relaxes the similarity comparison. This relaxation, eventually, activates a restart condition for the algorithm. Using our ancestry mechanism we implement a similar mechanism in our implementation of the algorithm for ordered chromosomes. In our case, relaxation is implemented by changing the depth of generations explored for similar ancestors. Once the number of generations drops below one (which just compares parents) we activate the same restart mechanism as CHC where the best individual is mutated to create a new restarted population. During this restart, we eliminate all ancestry information and completely start over.

4. Experimental Setup

With our inbreeding avoidance technique, our goal is to delay convergence using a single-threaded algorithm for as long as possible while generating good solutions to the problem. In the case of POPs, run-time is not, necessarily, the most important concern, and instead, the number of generations before convergence occurs is what we are hoping to extend in this work. To study if our inbreeding mechanism achieves this we will compare a GA and a preselection based GA to an implementation of the CHC algorithm and a preselection based GA with inbreeding prevention. Our comparison will be based on how many generations each of the algorithms generates before the best solution exists for 500 new generations. Additionally, we will experiment with the number of generations of ancestors to be recorded to see how this impacts the results.

Before showing data from the results of these experiments, we will describe some of the details for our TSP and each of the candidate GAs.

4.1 TSP instance

Instead of using a particular benchmark such as TSPLIB [21] we build our benchmarks with randomly created cities and the distances are based on Euclidean distance measurements. The reason for this approach is our need for large problems with a high number of nodes to look at POPs where the likelihood of finding a global optimum is unlikely. For each experiment we use the same benchmark to fairly compare each of the algorithmic approaches.

4.2 Common Algorithmic Parameters

To keep our experiments fair, there are a few parameters and operations that are common for all of the GAs in this work. Population size for all of the algorithms is set to 500 individuals per generation. The crossover operator is as described in section 2 and is the PMX based crossover. The mutation operator is a random swap between two locations in the genomic string and this value is set 5% of the number of cities in the TSP. The initial population for each of the algorithms is generated randomly.

4.3 Base GA

The base GA consists of previously described parameters and operators with the following additional aspects. The GA creates each new generation with approximately 20% of the population from crossbreeding, 79% from crossbreeding and mutations, and 1% random new individuals. Crossbreeding and mutations are taken from the best 25% of individuals in the previous population, and no parents are kept from one generation to the next. References to this algorithm will use the name "*base_ga*".

4.4 Preselection GA

Our implementation of the preselection GA has all the previously described common parameters. The crowding aspect of this algorithm is implemented based on the assumption

that children of parents are the most similar (without doing a formal comparison), and therefore, parents will compete directly with their children for the next generation. In our implementation, two parents are selected and the crossover operation is implemented. The two resulting children are then mutated and are grouped with the parents, and the best two individuals are propagated to the next generation.

In the case where inbreeding prevention is part of this algorithm, parents are only selected when they share no ancestry. If they do share ancestry then a new pairing is found by keeping one of the candidate parents and randomly finding a new second candidate parent. There may be a concern that depending on the depth of ancestry there will be no suitable pairing, but given the population size and depth of ancestors to be recorded, this does not occur for our implementation. However, algorithm designers should consider this when implementing a similar mechanism for their own problems.

References to these two algorithms will be "*preselect_ga*" and "*preselect_no_inbreeding_ga*" where the later has the mechanism to prevent inbreeding.

4.5 CHC algorithm

Our implementation of the CHC algorithm is derived from the original publication by Eschelman and a brief description is included for each of the 4 main concepts within his algorithm.

1) Parents and children combined together in competition for next population - This is implemented by ranking both children and parents together and then destroying the lower half of these individuals. In the case of a tie, the parent individuals are chosen first.
2) Inbreeding avoidance - this mechanism was described in the previous section, and when the population of individual remains the same from one generation to the next then the depth of ancestry search is reduced by one.
3) Highly disruptive crossover operator - as described in the background we make the assumption that the PMX crossover, where approximately 50% of the genetic material comes from each of the two parents, is sufficiently disruptive.
4) Full restart (from the best individual) once no new offspring are created in a generation - once the depth of search in the inbreeding avoidance mechanism reaches zero, we take the best individual and copy and mutate (with a 35% chromosome mutation rate) to create a new population. The algorithm then resumes normal operation.

References to this algorithm will be "*chc_algorithm*".

5. Experimental Results

In this section, we will look at two experiments. First, what happens to *chc_algorithm* as we change the number of past generations to record. These results will show us if there is any clear advantage to having a deeper record of ancestry. Next, we will look at how all three algorithms compare to each other observing how our mechanism improves the perseverance of diversity.

5.1 Impact of Ancestors on Diversity

For this experiment, we vary the depth of the ancestry tree between 3 and 8 generations for the *chc_algorithm*. In this experiment, the depth of ancestry search is controlled by the algorithm as described earlier in section 4.5. Each instantiation of the algorithms are executed until 500 iterations of the algorithm provide no improvement on the cost function and the run exits.

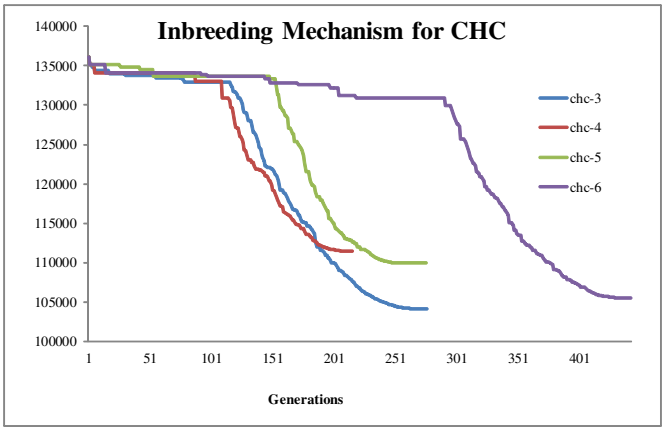

Fig. 5
SHOWS THE RESULTS FOR CHC WITH DIFFERENT ANCESTRY DEPTHS

Figure 5 shows a graph of our CHC implementation for the different ancestry depths. The y-axis shows the cost function measure for a TSP consisting of 2000 cities. The x-axis is the number of generations, where the last generation is reported before the 500 repetitions of no improvement. A legend is provided and the number for each corresponds to the maximum depth of ancestry to be recorded. Note that for each instance of the algorithm the random starting population is the same (chc-3, chc-4, chc-5, and chc-6 all have the same initial population).

From this example, we can see that as the ancestry depth is increased, the number of generations tends to increase. This tendency, however, is not the case for the 3 ancestor generation run labeled as *chc-3*. Also, the instance chc-3 finds one of the best solutions to the problem. This type of result is possible based on the randomness of the algorithm, and the more general result that diversity is maintained based on the trend that increasing the depth of generations recorded tends to increase the number of generations before the run exits.

To get a more thorough picture of what is happening, Figure 6 shows more runs of the CHC algorithm with inbreeding avoidance mechanism and a maximum ancestral depth of eight. For each of the five runs, we have colored the respective ancestry depth runs with the same coloring. The lower number of generations (3, 4, 5) are in dark colors, and the higher number of generations are in the lighter colors. The graph clearly shows that there is randomness for each run as expected. In terms of trends, the higher number of generation

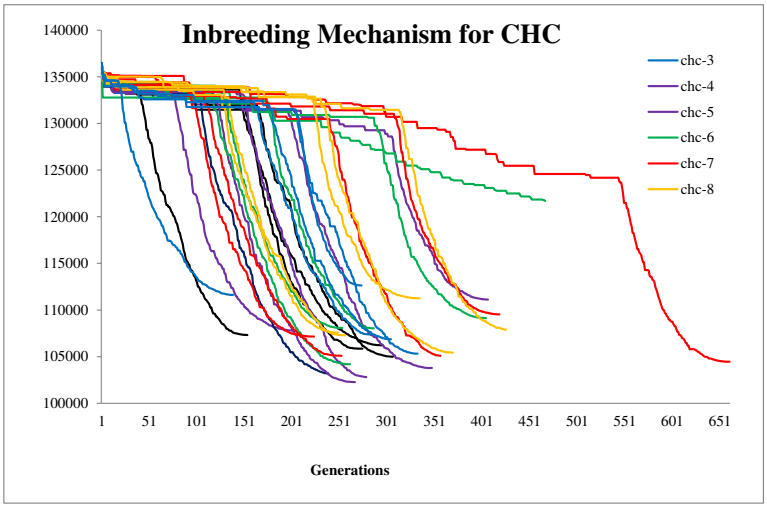

Fig. 6
SHOWS THE RESULTS FOR CHC WITH DIFFERENT ANCESTRY DEPTHS FOR 5 INSTANCES

based algorithms find a path to the low-energy solution later. The best solutions are found by execution runs by chc-5, chc-4, chc-6, and chc-7 in that order. We conclude that the depth range between 5 through 7 seems to be the best choice for ancestry records based on the results and the intuition that algorithms maintaining generations past 6 are unnecessarily restricting mating for unlike individuals.

One other thing we have observed in our CHC algorithm experiments is the lack of effectiveness for the restart mechanism. Only in 2 of the 30 runs of the algorithm was the restart effective in finding better solutions, noting that all 30 instances do initiate the restart mechanism. We hypothesize that random mutations for ordered chromosomes at low energy solution spaces is not effective, and other mechanisms need to be employed. We leave this to future work.

5.2 Comparison of All Algorithms

In this experiment, we compare all our implementations. Our comparison points include those algorithms with an inbreeding avoidance mechanism including the preselection algorithm with an ancestor depth of 6 previous generations. The exit condition is the same as previously described, and our overall goal is to maintain diversity as long as possible while finding good solutions.

Figure 7 shows a graph of our algorithmic implementations with a selection of CHC runs from the previous experiment. Similar to the previous graphs, the y-axis shows the cost function measure for a TSP consisting of 2000 cities, and the x-axis shows the number of generations, where the last generation is reported before the 500 repetitions of no improvement. The legend shows the name for each of the algorithm implementations with the numbering showing an algorithm with inbreeding prevention according to the depth of ancestors recorded. Note that the *preselect_no_inbreeding_ga* algorithm uses a depth of 6 ancestral generations to be recorded and when a pair of parents are selected, all ancestors are searched for common relations.

From the graph, we see a number of trends. First, preselection GAs maintain diversity for the greatest number of generations. This is to be expected since the crowding nature of this algorithm maintains highly diverse pockets of evolution. The best results generated by the preselection GAs, however, are not as good as the CHC algorithm. This is partially due to the pockets of evolution, which maintains diversity at a cost of less competition. The addition of the inbreeding avoidance mechanism in the preselection algorithm improves the quality of the solution and extends the diversity (number of generations), but not by a significant amount. For our purposes, the small crowding pockets do not seem to allow sufficient exploitation to find good solutions.

Overall, the best solutions are found by the more diverse algorithms that do not allow inbreeding based on the mechanism we have introduced. From our experimental data, this suggests that our inbreeding avoidance technique is providing the desired outcome, and overall, we observe significant improvement on diversity and quality for all the algorithms compared to our *base_ga*.

6. Conclusion

In this work, we introduced the concept of POPs and how GAs can play a valuable roll in solving these types of problems. We then explored how to maintain genetic diversity within a single-threaded GA run targeting POPs. We introduced an inbreeding avoidance technique inspired by tabu search, and we described how such a mechanism can be used both with the CHC algorithm and preselection algorithm for genomic strings that have ordered chromosomes. These types of chromosomes can be used to solve problems such as TSP and the FPGA placement problem.

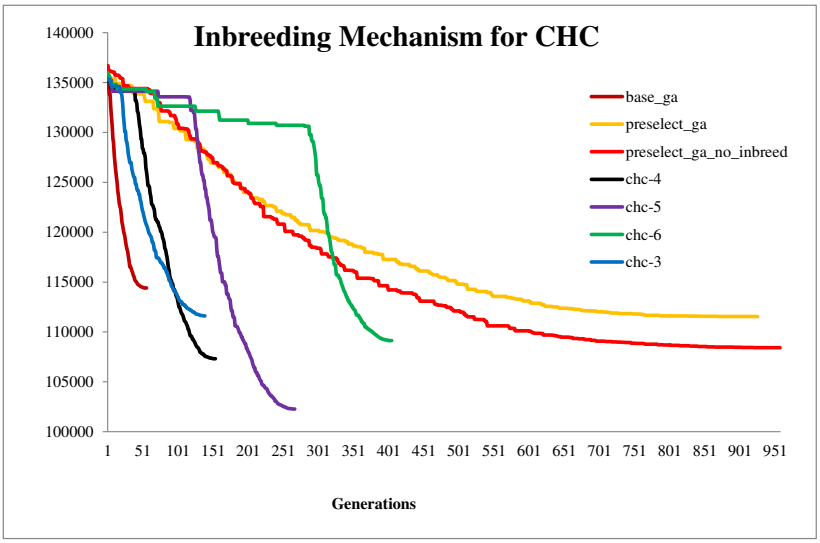

Fig. 7
SHOWS THE RESULTS ALL OUR THE IMPLEMENTATIONS.

The experiments with our inbreeding avoidance mechanism shows that the more levels of recording for generations improves the diversity of the population, and tends to improve the quality of results generated by the GAs. We believe that these types of mechanisms are not only valuable for POPs, but this could be exploited by other GAs for other types of problems.

References

[1] P. Jamieson, "Exploring inevitable convergence for a genetic algorithm persistent fpga placer," in *GEM*, 2011, pp. 1–8. [Online]. Available: http://www.users.muohio.edu/jamiespa/html_papers/gem_11.pdf

[2] ——, "Persistent cad for in-the-field power optimization," in *ERSA*, 2010, pp. 267–270. [Online]. Available: http://www.users.muohio.edu/jamiespa/html_papers/ersa_10.pdf

[3] M. Rocha and J. Neves, "Preventing premature convergence to local optima in genetic algorithms via random offspring generation," in *Proceedings of the 12th international conference on Industrial and engineering applications of artificial intelligence and expert systems: multiple approaches to intelligent systems*, 1999, pp. 127–136. [Online]. Available: http://portal.acm.org/citation.cfm?id=341506.341546

[4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. Addison-Wesley Professional, January 1989. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0201157675

[5] K. Roy and C. Sechen, "A timing driven n-way chip and multi-chip partitioner," in *Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on*, 1993, pp. 240 –247. [Online]. Available: http://www.eecg.toronto.edu/~jayar/pubs/sankar/fpga99sankar.pdf

[6] L. J. Eshelman, "The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination," in *FOGA*, 1990, pp. 265–283.

[7] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Comput. Oper. Res.*, vol. 13, pp. 533–549, May 1986. [Online]. Available: http://dl.acm.org/citation.cfm?id=15310.15311

[8] R. E. Smith, S. Forrest, and A. S. Perelson, "Searching for diverse, cooperative populations with genetic algorithms," *Evol. Comput.*, vol. 1, pp. 127–149, June 1993. [Online]. Available: http://dx.doi.org/10.1162/evco.1993.1.2.127

[9] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems." Ph.D. dissertation, Ann Arbor, MI, USA, 1975, aAI7609381.

[10] D. J. Cavicchio, "Adaptive Search Using Simulated Evolution," Ph.D. dissertation, University of Michigan, 1970.

[11] S. De, S. K. Pal, and A. Ghosh, "Genotypic and phenotypic assortative mating in genetic algorithm," *Inf. Sci.*, vol. 105, pp. 209–226, March 1998. [Online]. Available: http://dx.doi.org/10.1016/S0020-0255(97)10035-4

[12] R. Venkatraman and L. M. Patnaik, "An evolutionary approach to timing driven fpga placement," in *GLSVLSI '00: Proceedings of the 10th Great Lakes symposium on VLSI*, 2000, pp. 81–85. [Online]. Available: http://doi.acm.org/10.1145/330855.330986

[13] D. Whitley, D. Hains, and A. Howe, "A hybrid genetic algorithm for the traveling salesman problem using generalized partition crossover," in *Proceedings of the 11th international conference on Parallel problem solving from nature: Part I*, 2010, pp. 566–575. [Online]. Available: http://dl.acm.org/citation.cfm?id=1885031.1885092

[14] ——, "Tunneling between optima: partition crossover for the traveling salesman problem," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, 2009, pp. 915–922. [Online]. Available: http://doi.acm.org/10.1145/1569901.1570026

[15] D. E. Goldberg and R. Lingle, Jr., "Alleles, loci, and the traveling salesman problem," in *Proceedings of the 1st International Conference on Genetic Algorithms*, 1985, pp. 154–159. [Online]. Available: http://dl.acm.org/citation.cfm?id=645511.657095

[16] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the traveling salesman problem," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, 1987, pp. 224–230. [Online]. Available: http://dl.acm.org/citation.cfm?id=42512.42542

[17] B. A. Julstrom, "Very greedy crossover in a genetic algorithm for the traveling salesman problem," in *Proceedings of the 1995 ACM symposium on Applied computing*, 1995, pp. 324–328. [Online]. Available: http://doi.acm.org/10.1145/315891.316009

[18] V. A. Cicirello, "Non-wrapping order crossover: an order preserving crossover operator that respects absolute position," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pp. 1125–1132. [Online]. Available: http://doi.acm.org/10.1145/1143997.1144177

[19] L. Davis, "Applying adaptive algorithms to epistatic domains," in *IJCAI*, 1985, pp. 162–164.

[20] H. G. Barrow and R. M. Burstall, "Subgraph isomorphism, matching relational structures and maximal cliques," *Information Processing Letters*, vol. 4, pp. 83–84, 1976.

[21] G. Reinelt, "TSPLIB — a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.