Proceedings of Student/Faculty Research Day, CSIS, Pace University, May 4th, 2007

IMPROVING A GREEDY DNA MOTIF SEARCH USING A MULTIPLE GENOMIC SELF-ADAPTATING GENETIC ALGORITHM

Michael L. Gargano, mgargano@pace.edu
Computer Science, Pace University, NYC, NY 10038

Louis V. Quintas, lquintas@pace.edu
Mathematics, Pace University, NYC, NY 10038

Gregory A. Vaughn Sr., gregoryvaughn@verizon.net
Information Technology, St. Francis College, B'klyn, NY 11201

ABSTRACT

We consider the problem of combining a greedy motif search algorithm [16] with a self-adapting genetic algorithm employing multiple genomic representations in order to find high scoring substring patterns of size k in a set of t DNA sequences of size n thus improving the results of a stand-alone greedy motif search. We propose using multiple genomic (redundant) representations in a self-adapting genetic algorithm (GA) employing various codes with different locality properties. These encoding schemes insure feasibility after performing the operations of crossover and mutation and also ensure the feasibility of the initial randomly generated population (i.e., generation 0). The GAs applied in solving this problem employ non-locality or locality representations when appropriate (i.e., the GA adapts to its current search needs) which makes the GAs more efficient [15].

Keywords: DNA motif search, greedy, self-adapting genetic algorithm

Introduction to the Problem

We consider the problem of combining a greedy motif search algorithm with a self-adapting genetic algorithm employing multiple genomic representations in order to find high scoring substring patterns of size k in a set of t DNA sequences of size n thus improving the results of a stand-alone greedy motif search [16]. This will provide a greater robustness of search thus overcoming the problem with an individual greedy search.

In bioinformatics terminology, a DNA string of size n is simply a member of $\{ A, T, G, C \}^n$ and thus a word of length n consisting of letters from the alphabet $\sum = \{ A, T, G, C \}$.

Given a set of t DNA strings $d_1, d_2, d_3, \dots, d_t$ each of size n we wish to find positions s_i $(1 \le s_i \le n\text{-}k+1)$ in each of the t DNA strings d_i $(1 \le i \le t)$ so that the t substrings of length k each starting at position s_i in d_i will be as similar as possible. Given starting positions s_i in each of the t DNA strings we can take the letter C_{m+1} in the majority in each of the positions $s_i + m$ ($0 \le m \le k - 1$) and its corresponding count c_{m+1}. The resulting consensus substring $C_1, C_2, C_3, \ldots, C_k$, will represent a motif of size k for the substrings beginning at positions $s_1, \ldots, s_t$. We define the score of this motif to be: score = $c_1 + c_2 + c_3 + \ldots + c_k$. Notice that if there are ties the score is the same. A perfect score is tk.

The motif search problem is to find t substrings of length k each starting at position s_i in d_i that will provide the best score.

Example: Let t = 4	d_1	A T G C C G T A C C A T G G C
n = 15 and	d_2	T C A C G G A A T G C G C G T
k = 3	d_3	C C C T G A T G C T G G G C A
	d_4	G G C A T C C A T G C C G T A

The consensus for $s_1 = 3$, $s_2 = 1$, $s_3 = 2$, $s_4 = 12$ is CCT and the score is $2 + 3 + 2 = 7$. The consensus for $s_1 = 11$, $s_2 = 8$, $s_3 = 6$, $s_4 = 8$ is ATG and the score is a perfect $4 + 4 + 4 = 12$. It is unusual to find a perfect score in practice when n, t, and k are large.

A brute force search would require $(n - k + 1)^t$ scores to be calculated. Here we would consider each possible starting position for each DNA string.

Another approach which does not give very good results but needs only $(n - k + 1)^2 + (n - k + 1) (t - 2)$ scores to be calculated is a greedy method. Here we would consider each possible starting position for the first two DNA string to get an initial motif starting at s_1 and s_2. Then, one at a time, we would consider the next DNA string at step p and improve the best greedy motif based on the prior strings $s_1, s_2, s_3, \ldots s_{p-1}$. The effectiveness of this algorithm is somewhat dependent on the initial ordering of the given t DNA strings.

However, If we could find a very effective ordering from amongst the t! possible orderings of the DNA strings quickly the greedy algorithm would give an optimal/near optimal solution. Since finding such an ordering is NP hard we will solve it using genetic algorithmic methods.

Genetic Algorithm Methodology

A **genetic algorithm (GA)** is a biologically inspired, highly robust heuristic search procedure that can be used to find optimal (or near optimal) solutions to NP hard problems. The GA paradigm uses an adaptive methodology based on the ideas of Darwinian natural selection and genetic inheritance on a population of potential solutions. It employs the techniques of crossover (or mating), mutation, and survival of the fittest to generate new, typically fitter members of a population over a number of generations [1, 2, 3].

We propose GAs for solving this optimal sequencing problem using novel multiple genomic redundant encoding schemes. Our GAs create and evolve an encoded population of potential solutions so as to facilitate the creation of new *feasible* members by standard mating and mutation operations. (A feasible search space contains only members which satisfy the problem constraints, that is, a sequencing [4, 5, 6, 7, 13, 14].) When feasibility is not guaranteed, numerous methods for maintaining a feasible search space have been addressed in [11], but most are elaborate and complex. They include the use of problem-dependent genetic operators and specialized data structures, repairing or penalizing infeasible solutions, and the use of heuristics.) By making use of problem-specific encodings, our problem insures a *feasible* search space during the classical operations of crossover and mutation and, in addition, eliminates the need to screen during the generation of the initial population.

The Generic Genetic Algorithm

We now state the generic genetic algorithm we used for each application:

1) Randomly initialize a population of multiple genomic redundantly encoded potential solutions.
2) Map each population member to its equivalent phenome (ordering).
3) Calculate the fitness of any population member not yet evaluated.
4) Sort the members of the population in order of fitness.
5) Randomly select parents for mating and generate offspring using crossover.
6) Randomly select and clone members of the population to generate mutants.
7) Sort all the members of the expanded population in order of fitness adjusting each of the multiple segments to reflect the phenome with best fit.
8) Use the grim reaper to eliminate the population members with poor fitness.
9) If (termination criteria is met) then return best population member(s)
   else go to step 5.

Here a **population member** consists of an ordering of the t DNA strings. We adapted many of the standard GA techniques found in [1, 2, 3] to this problem. A brief description of these techniques follows. Selection of parents for mating involves randomly choosing one very fit member of the population and the other member randomly. The **reproductive process** is a simple crossover operation whereby two randomly selected parents are cut into sections at some randomly chosen positions and then have the parts of their encodings swapped to create two offspring (children). In our application the crossover operation produces an encoding for the offspring that have element values that always satisfy the position bounds (i.e., range constraints). **Mutation** is performed by randomly choosing a member of the population, cloning it, and then changing values in its encoding at randomly chosen positions subject to the range constraints for that position. A **grim reaper** mechanism replaces low scoring members in the population with newly created more fit offspring and mutants. Our **fitness** measure will be the consensus score using the greedy method using that ordering. **Termination** in the GA occurs when, for example, either a perfect score of tk is attained, no improvement in the best fitness value is observed for a number of generations, a certain number of generations have been examined, and/or a satisficing solution is attained (i.e., the result is not necessarily optimum, but is satisfactory).

Encodings

This application has multiple permutation encodings to identify the sequencing via different representations. Here we define the permutation code, forward code, and backward code for a permutation.

The 5-permutation 41532 or P[1] = 4, P[2] = 1, P[3] = 5, P[4] = 3, and P[5] = 2 can represent itself. This is one of multiple representations of 41532. We call this the **permutation code** and PC[1] = 4, PC[2] = 1, PC[3] = 5, PC[4] = 3, and PC[5] = 2.

An n permutation of the integers { 1, 2, …, n } can also be encoded by an array of size n where the value of the kth position can range over the values 1, 2, …, n-k+1.

An encoding of a permutation of the elements can also be represented as an array FC (**forward coding**) where $1 \le FC[k] \le n-k+1$ for $1 \le k \le n$. In order to decode a permutation code FC to obtain the permutation that it represents, begin with an empty array P of size n, then for $1 \le i \le n$ fill in the FC[i]th empty position (from left to right starting at position 1) of P with the value i.

Consider an example, with n = 5 and FC[1] = 2, FC[2] = 4, FC[3] = 3, FC[4] = 1, and FC[5] = 1 (or 24311) which represents the permutation P[1] = 4, P[2] = 1, P[3] = 5, P[4] = 3, and P[5] = 2 (or 41532).

Given a permutation array P, the reverse process begins with an empty array FC of size n, then for $1 \le i \le n$ starting with i = 1 and ending with i = n fill in the ith position of FC (from left to right starting at position 1) with the value k-(# of values $\le$ i that occur before position i in P) where P[k] contains the value i. (Note that FC[n] will always be 1, so that, we can shorten FC to an n – 1 element array if we wish.) An ordering of a set of 5 elements { e_1, e_2, e_3, e_4, e_5 } based on the forward code (24311) would then be 5-tuple (e_4, e_1, e_5, e_3, e_2).

An encoding of a permutation of the elements can also be represented as an array BC (**backward coding**) where $1 \le BC[k] \le n-k+1$ for $1 \le k \le n$. In order to decode a permutation code BC to obtain the permutation that it represents, begin with an empty array P of size n, then for $1 \le i \le n$ fill in the BC[i]th empty position (from left to right starting at position 1) of P with the value n-i+1.

Consider an example, with n = 5 and BC[1] = 3, BC[2] = 1, BC[3] = 2, BC[4] = 2, and BC[5] = 1 (or 31221) which represents the permutation P[1] = 4, P[2] = 1, P[3] = 5, P[4] = 3, and P[5] = 2 (or 41532).

Given a permutation array P, the reverse process begins with an empty array BC of size n, then for $1 \le i \le n$ starting with i = 1 and ending with i = n fill in the ith position of BC (from left to right starting at position 1) with the value k-(# of values $\ge$i that occur before position i in P) where P[k] contains the value n – i + 1. (Note that BC[n] will always be 1, thus, we can shorten BC to an n – 1 element array if we wish.) An ordering of a set of 5 elements { e_1, e_2, e_3, e_4, e_5 } based on the backward code (31221) would then be 5-tuple (e_4, e_1, e_5, e_3, e_2).

Next we consider a multiply redundant representation [12] that can be given by concatenating these lists. Thus a **multiply redundant representation** of the permutation 41532 would then be 243113122141532 with forward, backward, and permutation codes concatenated in that order. It is easy to mate and mutate this multiple representation scheme [6, 7, 13, 14], however the resulting list may not reflect the same phenotype in each segment of the multiple genome. In this case we simply choose a best performing segment and repair the entire multiple genome to mirror the best phenome in all of the other redundant segments.

Suppose	243113122141532	is a multiple genome for 41532
and	322211431152134	is a multiple genome for 52134.
Mating on positions	010110000110100	i.e.,swap positions 2,4,5,10,11,13
to get the child,	223213122151432	after swapping in those positions.

(Notice in last segment we get 51432 since positions 11 and 13 now reflect 4 and 5 in the first genome 41532 in the same order as the second genome 52134.)

Assuming the first segment 22321 represents the best phenome 51243, we repair the entire code and get 223211332151243 which is the multiple genome for 51243.

Conclusions

We considered using multiple genomic redundant representations in a self-adapting genetic algorithm to improve a greedy motif search algorithm in order to find high scoring string patterns of size k in a set of t DNA sequences of size n thus improving the results of a stand-alone greedy motif search for this NP hard problem. We then demonstrated that using multiple genomic redundant representations to create a self-adapting genetic algorithm by employing various codes with different locality properties that the genetic algorithm's efficiency was improved. The GA solving this NP hard problem, employs non-locality or locality representations when appropriate since the GA adapts to its current search needs making the GA more efficient.

Acknowledgements

We wish to thank Pace University's School of Computer Science and Information Systems and Dyson School of Arts and Sciences for partially supporting this research. We also wish to thank DIMACS at Rutgers University for workshops that introduced the authors to various bioinformatics problems.

References

[1] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, (2001).

[2] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, (1989).

[3] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, (1991).

[4] M. L. Gargano and S. C. Friederich, On Constructing a Spanning Tree with Optimal Sequencing, Congressus Numerantium 71, (1990) pp. 67-72.

[5] M. L. Gargano, L. V. Quintas and S. C. Friederich, Matroid Bases with Optimal Sequencing, Congressus Numerantium 82, (1991) pp. 65-77.

[6] M. L. Gargano and W. Edelson, A Genetic Algorithm Approach to Solving the Archaeology Seriation Problem, Congressus Numerantium 119, (1996) pp. 193-203.

[7] W. Edelson and M. L. Gargano, Minimal Edge-Ordered Spanning Trees Solved By a Genetic Algorithm with Feasible Search Space, Congressus Numerantium 135, (1998) pp. 37-45.

[8] F. S. Roberts, Discrete Mathematical Models, Prentice-Hall Inc., (1970).

[9] F. S. Hillier and G. J. Lieberman, Introduction to Operations Research, Holden-Day Inc. (1968).

[10] K. H. Rosen, Discrete Mathematics and Its Applications, Fourth Edition, Random House (1998).

[11] Z. Michalewicz, Heuristics for Evolutionary Computational Techniques, Journal of Heuristics, vol. 1, no. 2, (1996) pp. 596-597.

[12] F. Rothlauf and D.E.Goldberg,Redundant Representations in Evolutionary Computation, Evolutionary Computation, vol. 11, no. 4, 2003 pp.381-416.

[13] J. DeCicco, M.L. Gargano, W. Edelson, A Minimal Bidding Application (with slack times) Solved by a Genetic Algorithm Where Element Costs Are Time Dependent, GECCO, (2002).

[14] M.L. Gargano, W. Edelson, Optimally Sequenced Matroid Bases Solved By A Genetic Algorithm with Feasible Search Space Including a Variety of Applications, Congressus Numerantium 150, (2001) pp. 5-14.

[15] M.L. Gargano, Maheswara Prasad Kasinadhuni , Self-adaption in Genetic Algorithms using Multiple Genomic Redundant Representations, Congressus Numerantium 167, (2004) pp. 183-192.

[16] N.C. Jones, P.A. Pevzner, Bioinformatics Algorithms, MIT Press, (2004).