Rank Aggregation for Metasearch Engines using a Self-Adapting Genetic Algorithm with Multiple Genomic Representations

Michael L. Gargano
Computer Science, Pace University, NY
1 Martin Avenue
White Plains, NY 10606-1909
(011) 212 346 1687
mgargano@pace.edu

Maheswara P. Kasinadhuni*
Computer Science, Pace University, NY
1 Martin Avenue
White Plains, NY 10606-1909
(011) 732 977 8772
mpkasinadhuni@hotmail.com

ABSTRACT

In this paper, we consider the problem of combining rankings from the findings of various search engines in order to select documents based on differing and multiple criteria thus improving the results of a 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 NP hard 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: rank aggregation, metasearch, selfadapting genetic algorithm.

1. INTRODUCTION TO THE PROBLEM

We consider the problem of combining rankings from the findings of various search engines in order to select documents based on differing and multiple criteria thus improving the results of a search. This problem is concerned with finding a consensus ranking that "faithfully" represents and reflects the combined rankings of many different search engines. This will provide a greater robustness of a search overcoming individual search engine bias or other inadequacies.

Given a set of preference lists or rankings, we wish to find a consensus that minimizes the Kendall Tau distance (i.e., find a Kemeny optimal aggregation) [16]. The Kendall Tau distance between two rankings (i.e., n permutations) is the number of pairs of distinct integers a and b such that $1 \le a, b \le n$ where a and b are in the opposite order in each ranking. For example, if n = 6 and ranking #1 is 162345 while ranking #2 is 231645, we find the the Kendall Tau distance to be 4 since only the four pairs {1, 2}, {1, 3}, {2, 6}, and {3, 6} are in opposite order in each ranking. This is a standard method used by mathematicians to quantify the difference (i.e., distance) between the two rankings [16].

Here is a simple example of this problem. Suppose we receive the resulting priority lists (of size 5) from six different search engines and wish to form a consensus list using the distance method described above.

ranking #1	51324
ranking #2	34125
ranking #3	45312
ranking #4	14253
ranking #5	45321
ranking #6	43521

The consensus list that minimizes the Kendall Tau distance (i.e., finds a Kemeny optimal aggregation) is **45312** since the sum of the distances from each of the six rankings is 17 and is the smallest such total distance amongst all the possible lists (i.e., 5 permutations). A consensus is Condorcet if the ranking reflects the fact that if a is higher than b in a majority of the lists then a is ranked higher than b in the consensus [17]. However, since finding such a Condorcet consensus is not always possible, finding the minimium Kendall Tau distance (i.e., Kemeny optimal aggregation) is the best we can do while keeping in the spirit of the Condorcet idea [18].

Since finding such a consensus list is an NP hard we will solve it using genetic algorithmic methods.

2. GENETIC ALGORITHM METHOD

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 Non-deterministic Polynomial, 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.

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 (i.e., one with a small Kendall Tau distance) 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 Kendall Tau distance. The GA is terminated when, for example, either 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).

3. THE GENERIC GENETIC ALGORITHM

We can 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.
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.

4. 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 ≤ FC[k] ≤ n-k+1 for 1 ≤ k ≤ 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 ≤ i ≤ 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 ≤ i ≤ 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 ≤ 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 ≤ BC[k] ≤ n-k+1 for 1 ≤ k ≤ 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 ≤ i ≤ 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 $1 \leq i \leq n$ starting with i = 1 and ending with i = n fill in the i^{th} position of BC (from left to right starting at position 1) with the value k-(# of values ≥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.

5. RESULTS

The multiple genomic redundant representation **using all three segments was most efficient**. We experimented with all seven possible combinations, i.e., forward only, backward only, permutation only, forward&backward, forward&permutation backward&permutation and finally all three together forward&backward&permutation. The experiments using all three were consistently most efficient even though more overhead was generated.

The multiple genomic redundant representation using all three segments was also examined as to which representations dominated at various stages of the search. The **permutation code was used extensively in the earlier generations** when the GA was searching more globally since this coding scheme does not have a good locality property. **In the later stages the GA adapted its search using mostly the forward and backward codes** which have a stronger locality property.

6. CONCLUSIONS

We considered using multiple genomic redundant representations in a self-adapting genetic algorithm to solve the Kemeny optimal aggregation problem which is NP hard. 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, employ non-locality or locality representations when appropriate since the GA adapts to its current search needs making the GA more efficient.

7. ACKNOWLEDGEMENTS

We wish to thank Pace University's School of Computer Science and Information Systems for partially supporting this research. We also wish to thank DIMACS at Rutgers University for a workshop that introduced the authors to this interesting and important problem.

8. 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] C. Dwork,, R. Kumar, M. Naor, D. Sivakumar, Rank Aggregation Methods for the Web, ACM (5/2001) pp. 613-622.

[17] H.P. Young, Condorcet's Theory of Voting, Amer. Political Sci. Review, 82, pp.1231-1244,1988.

[18] H.P. Young, A. Levenglick, A Consistent Extension of Condorcet's Election Principle, SIAM J. Applied Math., 35(2), pp.285-300, 1978.