Ramsey Problem with Evolutionary Computation: Exploring Useful Crossover Operators

Robert Heckendorn, PhD.
University of Idaho
Computer Science
PO Box 441010
Moscow, Idaho
heckendo@uidaho.edu

Damian A. Ball
University of Idaho
Computer Science
PO Box 441010
Moscow, Idaho
ball7126@vandals.uidaho.edu

ABSTRACT

Evolutionary computation has had significant success in employing techniques that can navigate rugged and complicated solution terrain. Using the fundamental concepts behind natural evolution and number theory, EC scientists have revolutionized the way we search for the answers to difficult problems. We aim to apply these same concepts to yet another mathematical problem, Ramsey theory. By introducing a set of useful crossover operators and by using existing mathematical theory, we hope to create an EC application that can handle this NP-complete problem. This paper details the results of the operators we created.

General Terms

Ramsey theory

Keywords

Ramsey theory, evolutionary computing

1. INTRODUCTION

Ramsey theory is concerned with preserving characteristics of integer sets as those sets are seperated into distinct subsets, frequently called colors. More specifically, at what set size n can we guarantee, regardless of how the set is partitioned into some m subsets, certain characteristics will always exist in their respective subset. In the case of Ramsey, the set is comprised of all the edges in a complete graph. The characteristic we are preserving is a coloring of the edges such that we can guarantee that either a single color complete subgraph of size k_1 exists, or a k_2, or a k_3, up to k_n exists in its respective subset of edges, where $k_1, k_2, \ldots, k_m$ is a m-coloring of the graph. A common analogy for Ramsey is the party problem, which asks the minimum number of guests $R(k, l)$ that must be invited such that at least k people will know each other or at least l people will not know each other. In this paper we are concerned with 2-colorings, for which a Ramsey number is defined as $n = R(k, l)$. Additionally, for the 2-coloring Ramsey problem, a complete subgraph of size k is commonly refered to as a **clique** and l as an **anticlique**.

Ramsey theory has application in a wide variety of Computer Science and Mathematical subjects from communication complexity to computational geometry. Vera Rosta's paper [12] details recent advancements in Ramsey theory and their applications as an addendum to the book of Graham, Rothschild and Spencer [3]. Solutions to Ramsey $R(k, l)$ are frequently used in bounds for equations on other coloring problems [12].

Many values for $R(k, l)$ are still unknown. Stanislaw Radziszowski in [11] has maintained a table of known 2-color Ramsey numbers. $R(k, k)$ where $k > 4$ are still unknown. Bounds on $R(k, l)$ are usually found by either counter example or theoretical proof. We believe that evolutionary computation is a great framework to use for searching for lower bounds on Ramsey numbers by evolving counter examples.

2. BACKGROUND

Ramsey theory has been approached from a number of different perspectives over the past hundred years. Its NP-complete nature means it is particularly difficult to solve, yet recent advancements in computing have allowed scientists to try EC techniques to find solutions. [2, 4] are current methods that have been used to increase the bounds on certain Ramsey numbers.

Harri Haanpää's paper [4] describes the use of tabu local search on a random graph to find a new lower bound for R(5,9). Haanpää also partitions the edges into sets which have the same color. The edges are further partitioned into sets based on the distance ($dist(i, j)$) of their endpoints. Where $dist(i, j) = min(|i - j|, n - |i - j|)$ and n is the number of nodes. Using this technique, Haanpää was able to find 121 as the new lower bound for R(5,9).

In [2] Geoffrey Exoo uses tabu local search on random graphs along with simulated annealing to help the search settle as the algorithm approaches a solution. Exoo uses circle colorings of K_n as constructions. A circle coloring is where the color of an edge (i, j) is based on the result of $i - j$. Exoo was able to improve 13 Ramsey bounds using this method.

"Ramsey at home" [10] is a distributed computing project started in 2008 to find new bounds on Ramsey numbers. Their project narrative discusses using heuristics to help decrease the computation time associated with calculating Ramsey numbers. The distributed computation is used to quickly calculate smaller known Ramsey numbers and use the results to gain insight into their algorithms. The project has just launched its production code for deployment May 14th 2009.

Techniques not employed by the above papers include generic twoOp local search, computing on populations of candidate solutions, and any crossover operators. Thus, though local searching has been employed, we have not found an instance in which a full evolutionary computation algorithm was used to solve Ramsey numbers.

Creating effective crossover operators is key to the success of any EC application. We believe that using graph theory and other combinatorial mathematics related to Ramsey theory will allow us to create better crossover operators. These operators will allow our EC application to search the solution space more effectively and thus lead to solutions faster than random or brute combinatorial search.

In our Ramsey EC application we represent our solutions as edge matrices and store each candidate as an array of bit strings. For efficiency, we primarily perform operations on the upper triangular matrix and then adjust the lower to match. Our bit string manipulation functions are built with bitwise operators and are most efficient when running on a 64-bit processor. We have introduced three new operations to the existing Ramsey EC application.

2.1 mergeAwayFrom

The first is a crossover called mergeAwayFrom which takes two parents and produces a child whose edges are only modified if they are not shared by the two parents. This crossover can best be thought of as creating a line segment between the two parents and picking a point on the segment from which to create the child.

2.2 twoOp Local Search

The second is a two operation (twoOp) swap mutation operator. This is similar to many graph and EC problems where an edge or series of edges is modified in hopes of finding a better solution to the problem. This is also called local search.

2.3 twoOp with Simulated Annealing

The third is a modification to the twoOp operator that allows it to take swaps that produce worse fitness solutions. This ability is helpful when the solution space of the problem is rugged and can quickly lead a twoOp into a local minimum or maximum.

3. METHODS

For our experiments, the command-line inputs into the experiment program are our variables. We list the inputs in Table 1. Our focus for this paper is on $R(5,5)$ which currently has a lower bound of 43. Thus, our number of nodes variable is the same. Since we are interested in the subset of our input that deals directly with each experiment, we have kept other variables constant in addition to the number of nodes. The total combinatorial value of all variable inputs is listed in Table 2 by experiment. Experiment 1 did not include the TwoOp number variable.

Table 1: Experiment Variables

Variable	Value(s)
numNodes	43
popSize	2, 8, 16
maxevals	1000000
xoverselection	0, 1, 2
xoverprob	0, 0.3, 0.7, 1
xoversize	2
mutateprob	2
mutatesize	2
subgraphprob	1.0
mutatesubgraphprob	1.0
mergeprob	2.0, 8.0, 16.0
twoopnumber	10, 50, 100, 200

Table 2: Experiment Iterations

Experiment	Combinatorial value
1	39
2	156
3	156

3.1 Variables

The variables used in our experiments are described as follows: *xoverselection* indicates which xover operator to use if using xover. *xoverprob* is the probability of performing a xover and is checked each selection-insertion iteration. *xoversize* is used by the random xover operator to choose the amount of the candidates to xover. *mutateprob* is the probability of flipping a bit in the area selected for mutation. *mutatesize* is the probability of selecting a bit from the result of the mutation and placing it into the new candidate. *subgraphprob* is the probability of selecting a traditional xover instead of a semi-bipartite xover. *mutatesubgraphprob* is the probability of selecting a random mutation instead of a semi-bipartite mutation. *mergeprob* determines how much of the genetic material i, j do not share should be modified in j to make the new candidate. Finally, *twoopnumber* is the number of random edges in a candidate that are flipped during each selection-insertion iteration.

3.2 Fitness

We evaluated our fitness based on two ideas. One, we want to minimize our total number of cliques and anticliques. And two, we want our clique and anticlique distribution to be close to even. The following function places a greater weight on lowering the total cliques and anticliques. Figure 1 is a three-dimensional representation of the fitness function.

$$-\left[(cliques - anticliques)^2 + (cliques + anticliques)^2\right]$$

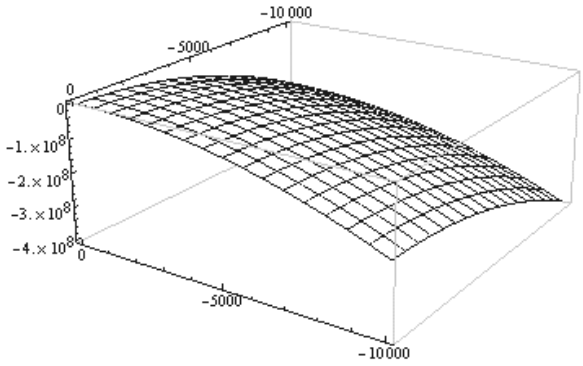

Figure 1: A three-dimensional graph of the fitness function.

3.3 Selection

Selection of candidates for crossover or mutation is made using the equation below. m is the probability of choosing candidates of worse fitness and n is a random number. For the purposes of our three experiments, m was kept constant at 0.5, giving an equal chance for picking a better or worse candidate for crossover or mutation.

$$popSize * (((\sqrt{m * m + 4 * (1 - m) * n}) - m)/(2 * (1 - m)))$$

$$0 \leq m \leq 1, 0 \leq n \leq 1$$

3.4 Insertion

Insertion was decided by fitness. If a new candidate was more fit than any single candidate in the existing population, then the new candidate was inserted into the population and the worse candidate in the population was removed.

Each experiment was run 50 times in order to collect a large enough sample size to deduce correlation. Some experiments took three hours for each complete combinatorial iteration. In order collect our results in a short period of time, we distributed the computation over fourteen dual-core machines. Results were calculated using standard deviation and variance to discover correlation between fitness and the operators we are experimenting on.

4. EXPERIMENTS

In this paper we explore three different combinations of crossover, local search, and random mutation. Each experiment runs for 1,000,000 evaluations. An evaluation is defined as an operation that calculates a portion or whole of a member of the population. Reading from an already calculated member variable is not considered an evaluation. Since we are trying to minimize a negative fitness, our insert function accepts fitness values that are less than the current population. Each experiment follows this EC algorithm.

The algorithm:

initialize *pop* (population) randomly
for each x in *pop*
  calculate fitness of pop_x
sort *pop* by fitness
loop (until solution is found or max evaluations is reached) :
  set i, j = distinct random *pop*
  if $xoverprob$! = 0 then
    if $xoverselect$ == 1 then
      xover of i, j
    else
      set i = mergeAwayFrom(j)
    insert i, j into *pop* if fitness of $i, j <$ some $pop_x \in pop$
  else
    set k = random mutation of i
  insert k into *pop* if fitness of $k <$ some $pop_x \in pop$

4.1 Crossover vs. Random Mutation

In this experiment we test the usefulness of our crossover operators against random mutation. For the xover selection variable, a value of 0 = no xover and thus strictly random mutation is used, 1 = random xover, 2 = mergeAwayFrom xover.

4.2 Local Search

In experiment 2 we introduce a new sub-algorithm to the existing algorithm. Every loop iteration we choose a number of edges to swap and randomly check to see if swapping them produces a population member with better fitness. We keep each swap if and only if it produces a better result.

4.3 Local Search with Simulated Annealing

In experiment 3 we introduce yet another sub-algorithm. In this case, every twoOp swap is evaluated against a simulated annealing function [8]. If the fitness is better, the option is always taken. In some cases however; if the solution is not better, the swap will still be kept. In order to use a simulated annealing function we introduced a temperature variable. The cooling schedule is built to exponentially reduce the temperature as the number of evaluations increases. Here is the simulated annealing function:

$$e^{\Delta D/T} > R(0, 1)$$

Where ΔD is the change of fitness, T is the current temperature, and $R(0, 1)$ is a random number from 0 to 1.

5. RESULTS

5.1 Experiment 1

The data we collected for experiment 1 indicates that our crossover operators are offering little to no benefit. Table 3 is an analysis of the fitness of our results. It highlights the variables which seem to directly affect our fitness. Since our fitness is negative and we are trying to minimize it, the mean fitness values start from numbers closest to zero. It is clear that the random mutation operator received the best fitness for all three population sizes. And thus, we need to rethink our crossover operators.

Table 3: Experiment 1 Results

pop	xover selection	mean	stddev
16	0	-46097	6343.884
8	0	-49262.68	6548.573
2	0	-54936.571	6542.905
16	1	-344657.04	419980.515
16	2	-357219.293	445502.795
8	1	-456636.6	590284.113
8	2	-478538.858	621571.261
2	2	-743790.259	1128622.605
2	1	-760415.306	1156801.849

5.2 Experiment 2

Based on our results from experiment 2, listed in Table 4, the twoOp operator is promising. While the improvement isn't drastic, it seems that the local search operation does better our fitness. It also appears that a larger population is still preferred along with the newly introduced twoOp evaluations variable. In opposition to experiment 1, xover selection doesn't seem to play as important a role. The removed data for Table 4 follows the trend of the data shown.

Table 4: Experiment 2 Results

pop	xover selection	twoOp number	mean	stddev
16	2	200	-43291.551	5205.931
16	0	200	-43648.455	5590.61
16	1	200	-44204.485	5295.836
8	0	100	-44932	5575.032
8	0	200	-44953.5	4300.952
16	0	50	-45050.273	5082.534
8	2	200	-45301.379	5542.678
16	2	100	-45792.288	5807.845
⋮	⋮	⋮	⋮	⋮
2	0	50	-52239.364	7160.235
8	1	10	-52526.439	7772.861
2	2	50	-52662.576	6933.963
2	0	10	-53652.955	6575.955
2	1	10	-54577.879	7620.191
2	2	10	-54953.404	6992.495

5.3 Experiment 3

Experiment 3 has the best improvement so far. Results are listed in Table 5. Much like experiment 2, population size and twoOp evaluations are the most influential variable on the fitness of our population. In comparison to experiment 1, we have smaller means and tighter standard deviations. We believe that this indicates an improvement in our local search operation. The removed data for Table 5 follows the trend of the data shown.

6. CONCLUSION

Based on data collected from our three experiments, we have concluded that more time and consideration should be given to our crossover operators. One of the defining characteristics of an evolutionary computation application, besides maintaining a population, is the crossover. Without it, we would be relying solely on random initialization and random change. Nevertheless, our local search operator was a success, especially when we applied simulated annealing.

Table 5: Experiment 3 Results

pop	xover selection	twoOp number	mean	stddev
16	2	200	-37974.476	4103.408
8	0	200	-38225.2	3985.09
8	2	200	-38771.982	4296.216
16	1	200	-39568.773	4185.916
16	0	200	-39666	4188.648
2	2	200	-40174.284	4755.192
8	1	200	-40341.307	4453.905
16	0	100	-40678.44	5572.624
⋮	⋮	⋮	⋮	⋮
16	1	10	-48069.627	6509.895
8	2	10	-48474.916	6099.021
8	1	10	-49207.973	5875.225
2	0	10	-50553.56	6935.306
2	1	10	-50773.507	5629.405
2	2	10	-51090.609	6633.559

For each iteration of our EC application we kept track of how many times we discarded a candidate. This number was used to generate a good-bad ratio which indicated for how many selection-insertion cycles our application spent generating unfit results. The iterations that solely used crossover generated some of the worst good-bad ratios out of the entire three experiments. Also, we made an indirect observation on computation time while running our experiments which has led us to strongly consider distributed computing for our next experiments. Some thoughts on how to improve our EC application are detailed in the next section.

7. FUTURE WORK

Throughout this semester we have been collecting a variety of ideas and papers that could help increase the efficiency of our EC application and/or help it converge to a solution. Those ideas range in subjects from evolutionary computation to graph theory. The few that look the most promising are listed below:

7.1 Simulated Annealing on Insertion

Since we had marked success with adding simulated annealing to our local search operation, there is a chance that we could see overall improvements with our application if we added simulated annealing to our insertion function. In this case, candidates whose fitness was not better than the entire population would still be allowed to enter the population on occasion.

7.2 Age-layered Population Structure

Gregory Hornby published [6] at the Genetic and Evolutionary Computation Conference in 2006. Hornby details a structure that tags each candidate with an age and divides the population into age-based layers. Candidates compete for survival strictly within their respective layers and are selected for breading in a similar restricted fashion. As the

candidate is used in crossover and mutation operations, its age is increased. Candidates created from one or more parents are given the initial age of the oldest parent. Every so often, the candidates in the lowest age-layer, the layer reserved for the youngest material, are randomized and given an age of 0. It is worth noting that this is a framework to help prevent premature convergence. It will not solve the ineffective crossover problem that we currently face. However, this framework does help an EC application navigate a rugged solution landscape.

7.3 Geometric Operators

In the paper [9], a geometric particle swarm optimization algorithm is described as a means to improve on existing particle swarm operators. A couple of paragraphs describe operators that construct sudo physics for crossover of genetic material. Considering geometric representations of our candidates and their genetic material could help us intelligently move about the solution landscape. The paper also introduces the notion of momentum of a candidate as it moves away from its initial position. This is basically a memory variable that the EC application can use during crossover.

7.4 Combinatorial and Graph Theory

Finally, [7, 1, 5] deserve particular attention because they have directly contributed to the most recent lower bounds on $R(5,5)$. Each paper describes combinatorial and graph theory concepts which were used to construct new lower bounds. Considering the math behind these papers could lead us to change the fundamental underlying representation of our candidates along with our operators. We leave these papers for future reading.

8. REFERENCES

[1] G. Exoo. A lower bound for r(5,5). *Journal of Graph Theory*, 13(1):97–98, 1989.

[2] G. Exoo. Some new ramsey colorings. *The Electronic Journal of Combinatorics*, 5(29):1–5, 1998.

[3] R. L. Graham, B. L. Rothschild, and J. H. Spencer. *Ramsey Theory*. Joel Wiley and Sons, New York, 1990.

[4] H. Haanpää. A lower bound for a ramsey number, 2000.

[5] D. Hanson. Sum-free sets and ramsey numbers. *Discrete Mathematics*, 14:57–61, 1976.

[6] G. S. Hornby. Alps: the age-layered population structure for reducing the problem of premature convergence. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 815–822, New York, NY, USA, 2006. ACM.

[7] R. Mathon. Lower bounds for ramsey numbers and association schemes. *Journal of Combinatorial Theory, Series B*, 42:122–127, 1987.

[8] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, and A. H. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087–1092, 1953.

[9] A. Moraglio and J. Togelius. Geometric particle swarm optimization for the sudoku puzzle. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 118–125, New York, NY, USA, 2007. ACM.

[10] N. Peterson. A heuristic approach towards finding ramsey numbers. http://www.ramseyathome.com/ramsey/, 2008.

[11] S. P. Radziszowski. A dynamic survey of small ramsey numbers. *The Electronic Journal of Combinatorics*, 1:1–60, 2006.

[12] V. Rosta. A dynamic survey of ramsey theory applications. *The Electronic Journal of Combinatorics*, 13:1–43, 2004.