An Incremental Approach to the Proportional GA

Han Yu
School of Computer Science
University of Central Florida
P. O. Box 162362
Orlando, FL 32816-2362, USA
hyu@cs.ucf.edu

Annie S. Wu
School of Computer Science
University of Central Florida
P. O. Box 162362
Orlando, FL 32816-2362, USA
aswu@cs.ucf.edu

ABSTRACT

The Proportional Genetic Algorithm (PGA) supports truly location independent solution encoding. In this paper, we propose a novel approach, called incremental building blocks, to improve the search performance of PGA. The main idea of this approach is to evolve low level building blocks individually in the beginning of a run and then continuously select and combine good building blocks to form larger and more complete ones until an optimal solution is found. Empirical studies in the resource allocation domain show that this approach is able to improve both the search quality and efficiency of a PGA. Further analysis of experimental data reveals that this approach is better at preserving the self-similarity of chromosomes during evolution.

Keywords

genetic algorithm, incremental building block, proportional genetic algorithm, resource allocation, genomic self-similarity

1. INTRODUCTION

The building block hypothesis states that solutions to a given optimization function can be evolved with a continuous process of recombining high-quality building blocks. While theoretical study has not given sufficient proof to this hypothesis, it is widely regarded to be the driving force for GA search. A GA typically starts with a randomized population of candidate solutions. A fitness function evaluates the quality of candidate solutions, and therefore, attempts to identify good building blocks from the current population. This process of building blocks identification, however, is somewhat inaccurate for two reasons. First, the fitness function evaluates the overall quality of a candidate solution, but is unable to measure the quality of individual pieces of a solution. Second, the quality of a building block may rely heavily on the context to which it belongs. Both the location of a building block in a solution and its correlation with other building blocks may affect the quality of the building block. As a result, the fitness function, in many cases, provides a fuzzy evaluation of the low level building blocks.

There have been some approaches to attack the above problem. Goldberg's messy GA [2] is one of the most well-known approaches. A messy GA begins with a population of partial solutions, or building blocks. The search for a solution is divided into two phases: a primordial selection phase which focuses on finding better building blocks from existing ones, and a juxtapositional phase which attempts to recombine building blocks to form complete solutions. Operators such as cut and splice are introduced in the juxtapositional phase to replace crossover. Messy GAs are shown to outperform simple GAs in deceptive functions. The success of messy GAs indicates that the idea of evolving and combining small building blocks may benefit the GA search.

We propose an approach called the incremental building blocks. This approach differs from traditional GAs in that the search for a solution actively attempts to optimize and combine building blocks. The search starts from a population of small building blocks. These building blocks are evolved over generations and good quality building blocks are selected for combination to form larger building blocks. We repeat this process until an optimized complete solution is found. We apply this approach to the proportional GA (PGA), in which the fitness is given based on the proportion of each type of gene in a chromosome rather than the gene ordering [3]. PGA has a truly location independent representation and has been applied successfully to domains such as resource allocation, symbolic regression, and control. To evolve low level building blocks separately, we need to be able to accurately estimate fitness of these building blocks. Since in PGA, building blocks are coarse representation of whole solution, the incremental building block approach should work perfectly. We evaluate the effectiveness of the incremental building block approach by measuring the performance of PGAs, both with and without the approach. We present the experimental results and provide a deeper analysis of the results.

2. ALGORITHM DESIGN

The main idea behind the incremental building block approach is to evolve low level building blocks separately and combine them periodically during the search for an optimal solution. This approach starts from a randomly generated population of size M consisting of short chromosomes or low level building blocks. In each generation, we evaluate fitness

of the chromosomes in the population, select better ones based on fitness, and mutate selected chromosome to form a new population of chromosomes. No recombination operator is used during the search. Chromosomes are combined in every *int* generations where *int* is a pre-specified parameter. The process of combination consists of four steps: 1) Randomly select two chromosomes as parents from the current population. 2) Combine the genetic code of the two chromosomes to form a new chromosome. We keep both the parents and the newly combined chromosome in the population. 3) Repeat the first two steps until we generate a population of M chromosomes. 4) Merge the parent and offspring populations by selecting the better chromosomes from both to form a new population of size M. The following pseudo code illustrates this approach.

```
Input - P0: initial set of M chromosomes,
      - int: the interval (number of generations)
        between two consecutive combination process
      - Gmax: maximum number of generations
Output - The best chromosome found in a run
Begin
Pcurr = P0;
For G=1 to Gmax, do
    If (G % int = 0), do
        /* combine building blocks every 'int'
        generations */
        Set Pnew to empty;
        For I = 1 to M, do
            Randomly select two chromosomes Bi and
            Bj from Pcurr;
            Combine Bi and Bj and insert the
            chromosome in Pnew;
        End for.
        /* combine both populations */
        Pcurr = Pcurr + Pnew;
    End if;
    Evaluate the fitness for all chromosomes in
    Pcurr;
    Select M chromosomes from Pcurr and form a new
    population Pnew;
    Mutate chromosomes in Pnew;
    Pcurr = Pnew;
End for.
Record the best chromosomes in a run as the final
solution.
End.
```

3. EMPIRICAL STUDY AND ANALYSIS OF RESULTS

3.1 Environment Design

We evaluate the performance of the proportional GA on the resource allocation function. Given a collection of resources, the goal is to evolve an optimal distribution. In our experiments, the optimal distribution is defined by a pre-specified target solution. A solution is encoded with a string of characters where each character represents a resource assigned to a resource recipient. The fitness is based on the proportion of each character in the entire string. A solution that encodes a resource distribution that is identical to the target distribution receives a maximum fitness of one. Equations 1 and 2 show the fitness function for a given chromosome c.

Parameters	Values
Population Size	400
Number of Generations	200
Crossover Rate	0.9
Mutation Rate	0.01
Selection	Tournament (2)
Combination Interval	5

Table 1: Parameter settings for the experiments.

$$f(c) = \frac{\sum_{n=1}^{N} Credit(n)}{N}, \tag{1}$$

where N is the number of resource recipients, and the function $Credit(n)$ evaluates the partial credits assigned to resource recipient n.

$$Credit(n) = \begin{cases} \frac{optimal(n)}{actual(n)} & \text{if } optimal(n) \leq actual(n) \\ \frac{actual(n)}{optimal(n)} & \text{otherwise} \end{cases} \tag{2}$$

where $optimal(n)$ and $actual(n)$ are the target and encoded proportions, respectively.

In this paper, we focus on the following problem. Empirical tests on variations of this problem produce consistent results. The problem consists of five recipients each expected to receive the following proportions of the resource: Recipient A: 4/47, Recipient B: 11/47, Recipient C: 18/47, Recipient D: 7/47, and Recipient E: 7/47. An optimal solution is a string of length 47 with the above proportions. We evaluate the effectiveness of the incremental building block approach by testing the PGA with and without this approach incorporated. We test fifty runs in each case with parameter settings listed in Table 1. When the incremental building block approach is used, we combine chromosomes every five generations. The crossover rate is only applied to traditional PGA runs. All other parameters are set as the same for both cases.

We set the initial size of chromosomes to be uniformly distributed between one and four for runs using incremental building block approach, and ten for traditional PGA runs. We do not introduce parsimony pressure specifically in the fitness function, as previous experiments have shown that the search performance of a PGA is very sensitive to the amount of parsimony pressure, which is somewhat domain dependent. Without parsimony pressure, PGA tends to evolve long chromosomes. We limit the evolvable chromosome length to ten times the length of an optimal solution (in this case, $10 \times 47 = 470$). When using the incremental build-

ing block approach, if the combination of two chromosomes exceeds the maximum allowed length, we do not combine the chromosomes and randomly select a parent chromosome to be the new chromosome. In traditional PGA runs, crossover is not allowed to occur if it produces offspring that exceed the maximum length. In that case, parents are copied unchanged into offspring.

3.2 Experimental Results

Table 2 gives our experimental results. Clearly, applying the incremental building block approach improves the search performance of PGA. Traditional PGA runs find an optimal solution only three times out of 50 runs, while the incremental building block approach is successful in eleven runs out of 50. Traditional PGA runs, however, produce a slightly high average fitness in terms of the best solutions found.

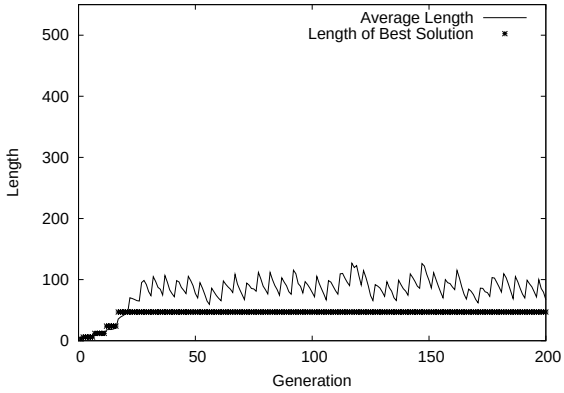
We also notice that the incremental building block approach tends to encourage GA to evolve shorter, more concise solutions, as exhibited by the average length of best solutions. Consequently, the execution time is significantly reduced. Closer examination of the best solutions in these runs reveals that the improvement of search efficiency is a direct result of the improved search performance. If an optimal solution is found in a run, it receives the highest fitness and has selection advantage over longer, less fit chromosomes. Hence the chromosome sizes can be conveniently controlled. If, however, an optimal solution cannot be found, longer solutions, which are capable of encoding more accurate resource proportions than shorter ones, generally receive higher fitness values and they quickly take over the whole population. Interestingly, we notice that all failed runs experience a sharp increase in chromosome size until the limitation is reached, no matter if the incremental building block approach is applied or not. Figures 1 and 2 show the changes in chromosome length during typical runs (a successful run and a failed run) of the PGA with and without the incremental building block approach, respectively.

Interestingly, experimental results indicate that, for both PGAs, early generations of a run is the deciding phase on the quality of solutions. Successful runs that find an optimal solution tend to find that solution within the first fifty generations. After this period, longer chromosomes have already dominated the population and finding an optimal solution (which is much shorter than the average chromosome length) from the existing population becomes very difficult, if not impossible. As we restrict the use of combination and crossover between a pair of long chromosomes, neither operator has any effect on the search process once a population has converged to near-maximum-length chromosomes. This unexpected side effect is a direct result of the length limitation and the restrictions on crossover and combination. In Section 3.4, we address this problem by modifying the PGA operators.

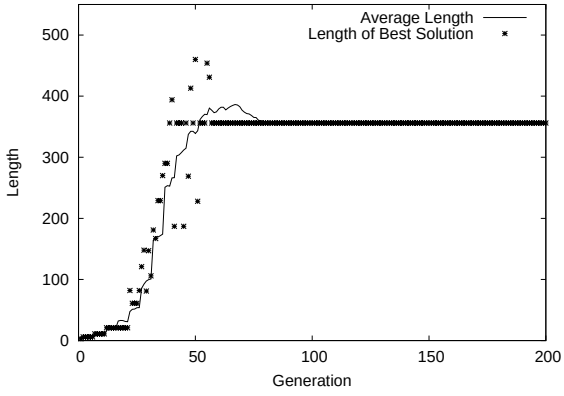
3.3 The Emergence of Self-Similarity

We attempt to give an explanation for why the incremental building block is able to improve the performance of PGA. We focus the study on the behavior of the emergence of self-similarity genes and hope to get a supportive answer.

Previous studies on the PGA indicate that, for location in-



(a)



(b)

Figure 1: The average chromosome length and the length of the best chromosome in the population for an example PGA run where the incremental building block approach is applied. (a) data from a successful run (b) data from a failed run.

Performance Metrics	PGA with Incremental BB	PGA
Number of Successful Runs	11	3
Fitness of Best Solutions (avg / std)	0.997 / 0.0017	0.999 / 0.0013
Length of Best Solutions (avg / std)	322.02 / 161.36	448.38 / 62.54
Execution Time in Seconds (avg / std)	29.38 / 10.15	41.80 / 0.27

Table 2: Comparison of experimental results for the two test cases.

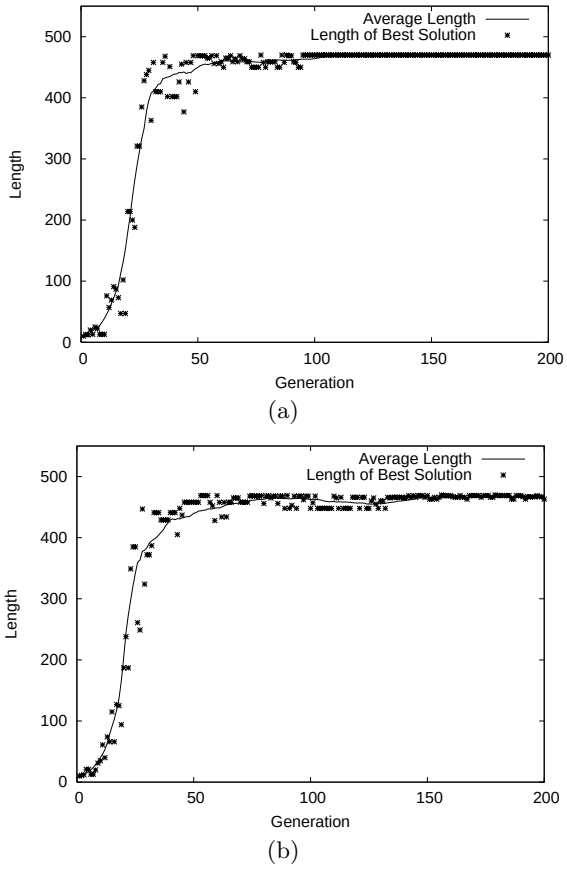


Figure 2: The average chromosome length and the length of the best chromosome in the population for an example PGA run where the incremental building block approach is not applied. (a) data from a successful run (b) data from a failed run.

dependent representation where the fitness is determined by the number of copies of genes rather than gene ordering, a GA tends to favor chromosomes that exhibit a high degree of self-similarity [1]. We extend this work to investigate whether the use of the incremental building block approach may affect the self-similarity of chromosomes.

We select the best solutions from runs in above experiments and calculate the self-similarity for each chromosome. If the length of a chromosome is shorter than the optimal solution, $l(optimal)$, the self-similarity value of that chromosome is the same as its fitness. Otherwise, the self-similarity is calculated as the average fitness of all segments of length $l(optimal)$ contained in the chromosome. Clearly, a truly self-similar chromosome has a self-similarity value of one. The lower the value, the less likely a chromosome exhibits self-similarity. Equation 3 shows the calculation of self-similarity for a given chromosome c.

$$Sim(c) = \begin{cases} f(c) & \text{if } l(c) \leq l(optimal) \\ \frac{\sum_{i=1}^{l(c)-l(optimal)+1} f(i)}{l(c)-l(optimal)+1} & \text{otherwise} \end{cases} \quad (3)$$

where $l(c)$ is the length of a chromosome c and $f(c)$ returns the fitness of a chromosome c.

Our statistical study indicates that the incremental building block approach enables a PGA to evolve more self-similar chromosomes. The average self-similarity of the best solutions over fifty runs is 0.901, which is much higher than those from traditional PGA runs (0.811). We speculate that the difference is due to the fact that the chromosomes in incremental building block runs are directly combined from shorter ones that already exhibit close resource proportions to the target solution. On the other hand, the crossover operator in PGA cannot accurately identify and combine the portions of chromosomes that contain close resource proportions between each other because the fitness function is evaluated on the chromosome level instead of on the portions to be combined. As a result, crossover does not guarantee to preserve the genetic self-similarity of chromosomes.

3.4 Further Improvement

The previous experiments indicate that it is very difficult for a PGA to find optimal solutions once the population becomes dominated by long chromosomes. In order to give PGAs more opportunity in finding optimal solutions, we

modify the genetic operators. In the incremental building block approach, we add a cutting operator. In the traditional PGA, we use truncation if crossover produces offspring that are too long.

Chromosome cutting is the reverse of combination and it is applied to single chromosomes. We randomly select a chromosome from the current population as the parent chromosome. A chromosome must contain at least two characters to qualify for the cutting operation. If the selected chromosome contains only one character, we randomly select another one until a qualified chromosome is found. We randomly select a cutting point within the parent chromosome and split the chromosome to two offspring. The first offspring inherits all characters to the left of the cutting point from its parent, and the second offspring inherits all characters to the right of the cutting point. Both the parent chromosome and its offspring remain in the population. We repeat the above operation until we have a population twice the size of the original population. Half of the population consists of the newly generated chromosomes from cutting and the other half consists of the original parent population. We evaluate the fitness of all chromosomes and select a new population with the same size as the original one. We apply the cutting operator in the same interval as the combination operator, and the two operators are performed alternately over a PGA run.

In the base PGA, crossover is forbidden if it produces an offspring that exceeds the maximum length. Truncation crossover simply truncates the excess portion beyond the maximum length and only retains the portion of a chromosome that does not exceed the limitation. Truncation crossover broadens the use of crossover and allows this recombination operator to produce shorter offspring than their parents, in spite of the size limitation imposed on the whole population.

We redo the previous experiments, applying chromosome cutting to the incremental building block approach and truncation crossover to the traditional PGA. The results, given in Table 3, show that both operators are very effective in improving the search performance. The incremental build blocks find an optimal solution in almost all runs (49 successful runs out of 50). The traditional PGA runs also succeed 12 times in 50 trials, much better than the previous results.

Figure 3 plots the changes in chromosome length in an example run of the incremental building block approach with chromosome cutting. An optimal solution is first found in generation 72 of this run. The average chromosome length increases in the beginning of the run, but decreases before reaching 300 and levels off for the rest of the run. This behavior contrasts with that shown in Figure 1(b) where the average chromosome length increases until it approaches the length limitation. Chromosome cutting is very effective in keeping the chromosome short throughout a run. As a result, a GA has more chance of finding an optimal solution whose length is much shorter than the length limitation. Out of all successful runs, 37 of them find the first optimal solution in a generation where chromosome cutting is applied. Once an optimal solution is found, it is more likely to survive in the population over generations.

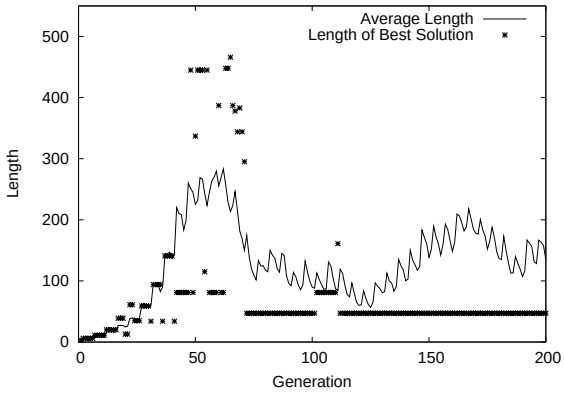


Figure 3: The average chromosome length and the length of the best chromosome in the population for an example PGA run where the incremental building block approach is applied and chromosome cutting is used.

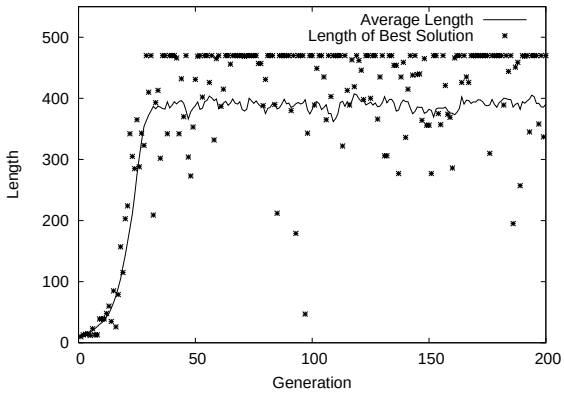


Figure 4: The average chromosome length and the length of the best chromosome in the population for an example PGA run where the incremental building block approach is not applied and truncation crossover is used.

Figure 4, which plots the changes in chromosome length in an example traditional PGA run with truncation crossover, exhibits similar behavior. Although the average chromosome length is still high, a PGA is able to evolve more concise and fitter solutions. In this specific run, the first optimal solution is found as late as generation 97.

4. CONCLUSIONS

In this paper, we introduce an approach called incremental building block to the proportional GA. With this approach, a PGA is initialized with a population of small building blocks and the search for a solution is a continuous process of evolving and combining good building blocks. Our experiments in a resource allocation domain show that the use of this approach is able to improve both the search performance and the efficiency of a PGA. This approach enables a PGA to evolve chromosomes with a higher degree of self-similarity. We also show that the performance of a PGA can be further improved by eliminating the following prob-

Performance Metrics	PGA with Incremental BB	PGA
Number of Successful Runs	49	12
Fitness of Best Solutions (avg / std)	1.0 / 0.0	0.993 / 0.0076
Length of Best Solutions (avg / std)	48.88 / 13.16	463.02 / 186.66
Execution Time in Seconds (avg / std)	21.42 / 3.84	43.91 / 0.36
Self-Similarity of Best Solutions (avg / std)	0.998 / 0.012	0.825 / 0.049

Table 3: A list of new experimental results when cutting and truncation crossover are used.

lem. Both the original PGA crossover and the combination operators become ineffective once a population converges to near maximum lengths.

The current status of this research offers ample opportunities for further exploration on this topic. We intend to apply the incremental building block approach to other domains in which a location independent representation can be used. We also hope to classify the domains where this approach is beneficial to the search performance of GAs.

5. REFERENCES

[1] I. I. Garibay, A. S. Wu, and O. O. Garibay. On favoring positive correlations between form and quality of candidate solutions via the emergence of genomic self-similarity. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2005.

[2] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[3] A. S. Wu and I. I. Garibay. The proportional genetic algorithm: Gene expression in a genetic algorithm. *Genetic Programming and Evolvable Machines*, 3(2):157–192, 2002.