Learning Building Block Structure from Crossover Failure

Zhenhua Li
School of Computer Science
China Univ. of Geosciences
Wuhan 430074, China
zhli@cug.edu.cn

Erik D. Goodman
Dept. of Electrical and Computer Engineering
2120 Engineering Building
Michigan State Univ., E. Lansing, MI 48824
goodman@egr.msu.edu

ABSTRACT

In the classical binary genetic algorithm, although crossover within a building block (BB) does not always cause a decrease in fitness, any decrease in fitness results from the destruction of some building blocks, in problems where such structures are well defined, such as those considered here. Those crossovers that cause both offspring to be worse, or one to be worse and one unchanged, are here designated as failed crossovers. Counting the failure frequency of single-point crossovers performed at each locus reveals something of the BB structure. Guided by the failure record, GA operators could choose appropriate points for crossover, in order to work more efficiently and effectively. Experiments on test functions Royal Road R1 and R2, Holland's Royal Road Challenge function and H-IFF functions show that such a guided operator improves performance. While many methods exist to discover building blocks, this "quick-and-dirty" method can sketch the linkage nearly "for free", requiring very little extra computation.

Categories and Subject Descriptors

I.2.m [**ARTIFICIAL INTELLIGENCE**]: Miscellaneous

General Terms

Algorithms, Performance

Keywords

Crossover disruption, building blocks, linkage discovery, genetic algorithms, "smart" operators

1. INTRODUCTION

The Building Block (BB) Hypothesis [9, 4], as confirmed by many theoretical studies [19, 1], set a roadmap for effective GA search for many types of problems: creating, growing and mixing BBs until finding a solution. Past practices for defining such a roadmap fall into two broad categories: linkage learning and probabilistic modeling [7]. Both fields are well researched and have generated many publications-for example, mGA [5] and LLGA [6] belong to the former, while the latter includes MIMIC [2], EDA [14] , cGA [8], BOA [16], HBOA [11, 15], etc. These approaches are typically complex to implement relative to a "normal" GA.

Here, for problems exhibiting a strong non-overlapped building block structure, we present a simple, although admittedly less powerful, alternative for exploring the linkage, by tracking crossover disruption at each position in the genome. We know that although crossover within a building block (BB) won't always cause a fitness loss in a BB-structured problem (i.e., one in which epistasis is not significant-there are minimal nonlinear dependencies among loci that are widely separated on the chromosome), when crossover DOES produce a decrease in fitness of both offspring, it will be because a building block, right on the crossover point, was destroyed. Those crossovers that cause both offspring to be worse, or one worse and one unchanged, are designated here as failed crossovers. By counting the failure frequency at each position on the genome, one can gain information about the BB structure. This information can supply appropriate points to operators. For example, positions where no failures occur are good candidates for future recombination to increase the length of building blocks, rather than disrupting them.

The method works like an adaptive operator, but differs from others developed earlier in its involvement of building block concepts, and therefore, ability to reflect genome structure. A related operator [17] was designed to be adaptive by recording points at which successful crossovers were done and then following them by other crossovers at the same points. In contrast, we use a different criterion-points of failure (both offspring being worse)-which provides even more absolute guidance, and apply it in determining future crossovers. The work also is different from the rule induction GA [18] or machine learning evolution [12]. Although both of these authors also had interest in crossover points yielding fitness decreases, their work did not fully capture the concept of building block destruction as occurring when crossover at a given position produces two worse offspring (or one worse and one unchanged), and therefore probably couldn't explore the BB structure effectively.

This paper first reviews the BB theory in a hierarchical view, and based on which, why and what types of operator outcomes could be an indication of a tight non-overlapped BB is discussed, then a Genetic Algorithm is designed to track crossover failure for guiding future operators. Finally,the

GECCO'07 July 7-11, 2007, London, England, United Kingdom.

guided GA is comparing with the unguided one on four test functions. Conclusions are drawn at the end.

2. THE BUILDING BLOCK HYPOTHESIS: A HIERARCHICAL VIEW

The Building Block Hypothesis supposes that short, low-order, high-fitness blocks can be combined into higher-level ones and finally into a solution. This idea is illustrated in Figure 1.

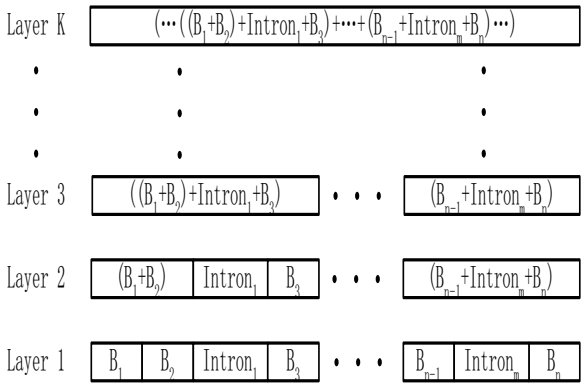

Figure 1: Hierarchical Organization of Building Blocks

Figure 1 illustrates a pyramid (or tree) structure, where each higher-level BB is composed of level-lower BB(s) and intron(s), for example, the leftmost BB in level 2 consists of two BBs from level 1, and also is a part of a BB of a higher level. This higher-level BB can be expressed as: $((B_1+B_2)+Intron_1+B_3)$. Note that once the intron is incorporated into a higher-level building block, it is no longer an intron at the higher level.

From the point of view of an evolutionary process, the levels represent different stages. A lower level tends to be assembled at an earlier time in the GA's evolutionary process. Because lower-level building blocks are discovered before higher-level ones of which they are components, lower-level BBs have a larger chance to be disrupted by crossover, when viewed over time. That will be illustrated in the following experiments.

3. OPERATIONS AND BUILDING BLOCK DISCOVERY

Here, one-point crossover and single-bit mutation are compared on exploring tight non-overlapped BB structure.

3.0.1 Why Not Track Mutations

Though results of one-bit mutation could also be used to detect some BB structure, distinguishing an intron from a BB by observing whether there is a fitness change after a mutation at a particular site, it does not work well in the following situations.

1. In hierarchical assembly of BBs, it can only find the lowest-level BB structures. In Figure 1, it works on level 1 only and fails to identify the assembled BBs at higher levels. To the mutation operator, a lowest-level intron always appears to be an intron at all levels, since mutating it never results in a fitness change, even though the intron may have become part of an assembled BB at a higher level in later the evolutionary process.

2. It cannot distinguish neighboring BBs, though can explore adjacent BBs and introns. For example, given an 8-bit string where the first half is a BB and the second half is an intron, single-bit mutation could easily get a result expressed as "YYYYNNNN" (where Y means there is a fitness change after 1-bit mutation, N means not), but it will give the result "YYYYYYYY" for two adjacent 4-bit BBs, obscuring the BB boundary.

3.0.2 Why Track Crossover Failure?

Comparing to tracking single-bit mutations, tracking one-point crossovers is capable of identifying BBs at different levels and distinguishing the BB boundaries easily, although not all crossover operations yield good indicators.

Of the various classes of crossover outcomes, "failed" crossovers–i.e., those resulting in a fitness decrease in both offspring (or decrease in one and no change in the other)–are the most informative. It is assumed that they are destroying a BB, so are not occurring within an intron, since, by definition, changing the components of the intron would not affect fitness, and no building blocks would be disrupted. Thus, "failed" crossovers are taking place within BBs (although the converse does not hold–i.e., recombination occurring within a BB does not always cause a fitness loss). This observation provides a way of identifying linkage–namely, by tracking the frequency of crossover failure at each position on the chromosome. Given a set of "bad" crossover positions (i.e., sites at which crossover produced failures), the crossover operator should be adapted so as to avoid them, choosing instead to crossover between BBs and increasing the probability of hierarchically assembling them.

3.0.3 About Tracking Crossover Success

While fitness loss in both offspring after one-point crossover indicates that BB destruction has occurred at the crossover point, fitness increase in both offspring also implies BB construction.

However, crossover success, defined as both offspring having better fitness or one better and one not worse after one-point crossover, occurs much less frequently than crossover failures. To create a new BB, all components must be perfectly assembled, but to destroy a BB can be done in many ways. For example, to form a new 8-bit BB, "11111111", requires assembling exactly 8 ones; while to break up an existing BB requires only a 0 in any of those positions. The experiments reported here on Royal Road R1 and R2 functions show that the frequency of crossover success on R1 is about 1/35 the frequency of crossover failure, and on R2, about 1/20 ,based on 100 independent runs (the ratio will increase if we use guided points to avoid crossover disruption, to significantly decrease the number of failed crossovers). It might be better track crossover success as well as failure; however, this has not been done in the work reported here.

In summary, crossover failure within BBs occurs with high frequency compared to crossover success; most crossovers internal to a BB are likely to cause a fitness loss (destroying an exist BB) rather than benefit (producing a new BB), especially late in the evolutionary process, when lower-level BBs have already been produced and the challenge is to assemble higher-level BB combinations.

4. EXPERIMENTS ON R1, R2, RR AND H-IFF

4.1 Experimental Design

4.1.1 Test Examples

Royal Road functions R1 and R2, Holland's Royal Road Challenge function and the H-IFF function are chosen to be tested step-by-step, representing pure single-level BBs only, BBs with a hierarchical structure, hierarchical BBs with deception and hierarchical BBs with multiple optima, respectively.

Royal Road R1 and R2.

Functions R1 and R2 are designed as strings composed of several BBs [13, 3]. R1 is flat, expressed as
(B1,B2,B3,B4,B5,B6,B7,B8)
(see Figure 2[3]), while R2 is hierarchical, expressed as:
(((B1,B2), (B3, B4)), ((B5, B6) , (B7, B8)))
(see Figure 3[3]).

s_1=11111111**********;c_1=8
s_2=********11111111**********;c_2=8
s_3=****************11111111**********;c_3=8
s_4=************************11111111**********;c_4=8
s_5=********************************11111111**********;c_5=8
s_6=**11111111**********;c_6=8
s_7=**11111111**********;c_7=8
s_8=**11111111;c_8=8
s_{opt}=1111111111111111111111111111111111111111111111111111111111111111

Figure 2: An optimal string broken up into 8 building blocks. The function R1(x) (where x is a bit string) is computed by summing the coefficients c_s corresponding to each of the given schemata of which x is an instance. For example, R1(1111111100...0) = 8, and R1(1111111100...011111111) = 16. Here c_s = order(s).

s_1=11111111**********;c_1=8
s_2=********11111111**********;c_2=8
s_3=****************11111111**********;c_3=8
s_4=************************11111111**********;c_4=8
s_5=********************************11111111**********;c_5=8
s_6=**11111111**********;c_6=8
s_7=**11111111********;c_7=8
s_8=**11111111;c_8=8
s_9=1111111111111111**********;c_9=16
s_{10}=****************1111111111111111**********;c_{10}=16
s_{11}=********************************1111111111111111**********;c_{11}=16
s_{12}=**11111111111111;c_{12}=16
s_{13}=11111111111111111111111111111111**********;c_{13}=32
s_{14}=********************************1111111111111111111111111111111;c_{14}=32
s_{opt}=1111111111111111111111111111111111111111111111111111111111111111

Figure 3: Royal Road Function R2. R2(x) is computed in the same way as R1: by summing the coefficients c_s corresponding to each of the given schemata of which x is an instance. For example, R2(1111111100...011111111) = 16, but R2(1111111111111111100...0) = 32, and R2(11111111...1) = 192.

Originally, R1 and R2 were designed to investigate how the "stepping stones"–intermediate-order schemata–affect the GA performance. Here, they are taken as test examples with a flat BB structure and tree BB structure, respectively, to see whether the failed crossover can find the structure, rather than to study the GA's behavior, as the initial researchers did.

Holland's Royal Road Challenge.

The challenge, RR(JH), presented by John Holland at ICGA '93, then posted on GA-List, is to test a GA's performance on BB discovery and combination. The function is

$$Score = \sum_j BONUS(j) + \sum_i PART(i);$$

where j indexes levels in the hierarchy (1 is the lowest level); i indexes target schemata (1 is at left). There are 2^k target schemata at level 1, and 2^{k-j} target schemata at level j+1 (compounded of adjacent pairs of schemata from the next lower level); each target schema is defined over b (blocksize) loci. Here,
genome length = (blocksize+gapsize) * 2^k.

$$BONUS(j) = \begin{cases} u^* + (n(j)-1)u & , \quad j > 0 \\ 0 & , \quad j = 0 \end{cases}$$

where n(j) is the number of targets found at level j and u^* and u are parameters, $u^* > u$.

$$PART(i) = \begin{cases} m(i)v & , \quad if\ m(i) < m^* + 1 \\ -m(m(i) - m^*)v & , \quad if\ m^* < m(i) < b \\ 0 & , \quad otherwise \end{cases}$$

PART(i) is the contribution to the overall score from m(i) correct alleles in target schema i at the lowest level, $0 < i < 1 + 2^k$. RR(JH) is a hierarchical function with deception. In the following experiment, the parameter setting is string length=240 bits, $b = 8, k = 4, u^* = 1, u = 0.3, v = 0.02, and\ m^* = 4$, with 7 bits of intron between every two blocks.

Hierarchical-if-and-only-if (H-IFF).

H-IFF is a recursive function for defining hierarchically consistent decomposable fitness functions from two base functions, F(B) and T(B) [20, 21]. T(B) evaluates the relationship of BBs, and F(B) gives the score according to both T(B) and the score of BBs in the next level-for example, for a four-variable problem with blocks:
F(a,b,c,d) = 4f(t(t(a,b),t(c,d))) + 2f(t(a,b)) + 2f(t(c,d)) + f(a) + f(b) + f(c) + f(d). [20]

Therefore, the higher-level BB is credited only if its lower-level blocks are mutually consistent (all 1's or all 0's). For instance, although "0000" and "1111" are good building blocks alone, their combination "00001111" gets no bonus. The Hamming Distance between the solutions that are of second-highest fitness and the global optima (of which there are also two) is greater than the distances between any other pair of local optima, and the differences in fitness maintain the same relationships. Such a feature makes it is easy for a search algorithm to become trapped in local optima, since it is hard to replace an entire (non-globally-optimal) part of a solution accurately to achieve the best result. This is a form of deception, and helps to make the H-IFF problem more challenging. Here, a 64-bit H-IFF problem is studied.

4.1.2 Algorithm

Rather than using a traditional GA's way of generating the next generation using serial application of operators (selection, crossover and mutation), we will instead use GP's

"parallel" method, in which only one operation is performed in generation of each offspring (or pair of offspring, through crossover), rather than allowing, for example, mutation also to act on an offspring just created by crossover.

Such a method is fit for the purpose of observing the effect of crossovers. Tracking crossover failures requires evaluating the new individuals right after the crossover (without possible intervening mutation), to see whether or not the crossover has "failed"; if so, the crossover point is tallied as an internal BB point, to guide future operations. If we let a mutation follow crossover and then evaluated the result, as in a classical GA, it would confound the effects of the operators, making it hard to distinguish effects of crossover from those of both crossover and mutation.

The Algorithm is.

```
Initialize population;
Evaluate population;
WHILE not done
  CASE Operation OF
  Select for survival:
      Select one individual for survival
      unchanged to the next generation
  Crossover:
      Select two individuals
      Crossover them w/wo suggested points
      Evaluate them
      Record whether or not the crossover point
      as a internal BB point
  Mutation:
      Select one individual
      Mutate
      Evaluate it
END WHILE
```

4.1.3 GA Setting

To simplify the experiment, the operators are one-point crossover and single-bit mutation only, and a large population size is used, in order to increase the similarity of the runs. In addition to the normal GA parameters, two variables are employed as follows:

1. Sampling Generations (SG), by which time nearly all loci are expected to have been well tested by crossover. Prior to this generation, it is supposed that the BB structure has not been sufficiently explored to put an end to exploration and simply exploit the information to guide future operations. After SG, it is supposed that the tracking has been sufficient to constitute a reliable foundation for guiding future operations.

   However, even the data gathered prior to SG can be used to guide crossover to avoid many useless failures, even though it is not complete. It is feasible to allow some crossovers to be guided by the existing data prior to SG, and to allow others to continue to explore the space of crossover points until the crossover data are deemed reliable. In the experiments reported here, a measure Current Generation (CG) / Sampling Generations is defined, and is used to decide what crossover percentage is to be guided by the data gathered to date. At early stages, CG is small, and since SG is a larger constant, the quotient is small and few crossovers are guided. As CG approaches SG, the percentage of guided crossovers increases. After CG >= SG, all crossovers are guided.

   Generally, determination of SG depends on genome length and the complexity of the problem. In our experiments, we used SG as a constant 10. However, an adaptive SG, based on statistical analysis of crossover history, is planned for future work.

2. Threshold parameter T, used to determine a "failed" crossover to be used for future guidance. Since potentially all points could be crossover failure points, even points in an intron (if it is a part of a higher-level BB), we must focus crossovers at those points exhibiting more failures, but not at all points exhibiting ANY failures. In our experiments, those points with failure frequencies less than the half of the average are rejected as likely BB boundaries or introns in higher level.

Table 1: GA Parameters

	R1	R2	RR(JH)	HIFF
Crossover	one-point crossover			
Mutation	single-bit mutation			
Selection	tournament, size 4			
Pop. size	2000		5000	
P Selection	0.2			
P Crossover	0.7			
P Mutation	0.1			
No. of Gens.	50			
No. of Runs	100			
Sampling Gens.	10			
T	0.5*average failure points			

4.2 Results

4.2.1 Unguided GA(U-GA) vs GA Guided by Failed Crossovers (G-GA)

Based on 100 runs on each test function stated above, a performance comparison between and U-GA and G-GA is provided in following sections.

Evaluations.

Each test condition was run 100 times independently, and the results are shown in Table 2.

Table 2: Average Evaluations of U-GA and G-GA to find global optimum in 100 runs. (T-test: unequal variance, independent, one-tailed)

	GA Type	Success Rate	Mean	Std Dev	T-test
R1	U-GA	100%	30925	2411	8.7
	G-GA	100%	25756	1733	
R2	U-GA	100%	29731	2971	7.8
	G-GA	100%	24335	1707	
RR(JH)	U-GA	100%	107312	10016	5.2
	G-GA	100%	94376	7464	
H-IFF	U-GA	95%	77593	9628	6.1
	G-GA	99%	64307	4371	

Clearly, the G-GA significantly outperforms U-GA on all four experiments both in speed and robustness

(T(v=100,P_r=0.9995)=3.39)). The decreases in average numbers of evaluations are 17%,18%,12%,17% on R1,R2,RR(JH) and H-IFF, respectively. And the less standard deviation of G-GA in all experiments shows that it is more stable than U-GA. (It is interesting that we did achieve the results anticipated by the proposers of R1 and R2 (i.e., R2 was quicker to solve than R1), and showed that the "stepping stones" worked, while they didn't get those results in their experiments. That difference may be because of the large population sizes used here, which can help to alleviate the effects of hitchhiking, as suggested by [10].)

Crossover Failure Frequency.

Why G-GA is faster than U-GA is that G-GA experiences fewer crossover failures, and therefore its crossover works more efficiently and effectively.

The following charts demonstrate the comparison of U-GA and G-GA on aggregate crossover failures, versus generation, for the four functions above, over 100 runs. Since we use GP's "parallel" method to produce the next generation, the percentage of each operation(selection,crossover, mutation)in each generation is fixed, so, the evaluations on each generation is nearly same based on same GA setting, that is to say, the generations on Figure 4 and Figure 5 could be seen as an equivalent of evaluations.

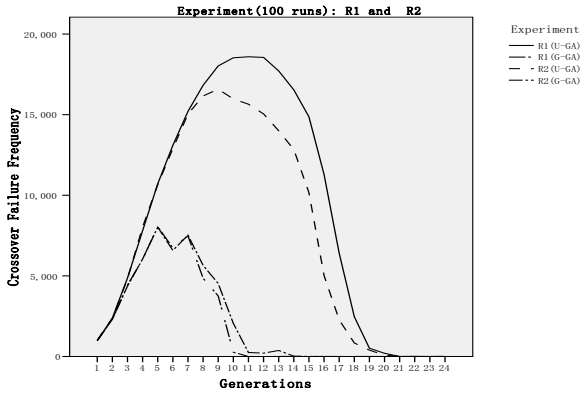

Figure 4: Accumulated crossover failures at each generations of R1 and R2 over 100 runs(Population Size: 2000)

Both Figure 4 and Figure 5 indicate the G-GA does many fewer "bad" crossovers than U-GA. The tailing off of all curves is because each of the 100 runs is stopped when it finds the global optimum value, and for future generations, its number of crossover failures (and of operations) is averaged in as zero.

At the beginning, the crossover failures of all G-GAs are the same as for their counterparts, but after that, their failure frequencies are always smaller than those of U-GAs, though both of their frequencies go up on the following generations.

Then G-GA failures reach their peaks at about the 5-th generation, which is when the BB-structure-learning-oriented crossover begins to be exceeded by the BB-structure-utilization-oriented crossover (since we set the 10th generation as the Sample Generation parameter SG), and the peaks are half of those of their counterparts, showing that only 50% crossover

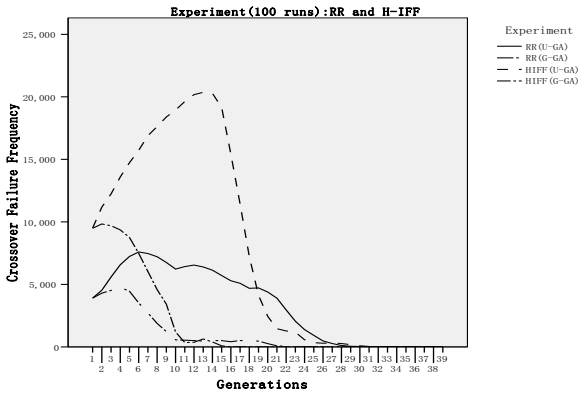

Figure 5: Accumulated crossover failures at each generations of RR(JH) and H-IFF over 100 runs(Population Size: 5000)

on BB detection and get failures, while the other 50% crossover guided don't, right on this generation.

Finally the G-GA failures begin to bottom out at about the 10th generation, after which BB combination, rather than BB production or BB destruction, is the only task for the G-GA. However, there are still a few failures after SG; the reason is that we set a non-zero threshold T on the failure values, below which we still perform crossovers (assuming that those positions are not between building blocks). Therefore, crossovers will still be performed at these positions, and probably produce some failures.

Among the four functions above, the first three functions, R1,R2 and RR (JH), have similar variation between G-GA and U-GA, but for the last one, H-IFF, the G-GA does not "jump" during its first five generations as U-GA does. We guess that since each peak of G-GA failures is half of U-GAs, but on the H-IFF, it's start is already aprroachs the peak, so there is no room for a jump from its start to the 5th generation, the G-GA failure peak.

4.2.2 BB Structure Discovered by Crossover Failure

For purposes of illustration, we have chosen the first run of each G-GA 100-run set to produce the following charts, to see how the crossover failures present the BB structure. The accumulated tracked failures over all 100 runs are also rendered, showing an even clearer description of the BB structures.

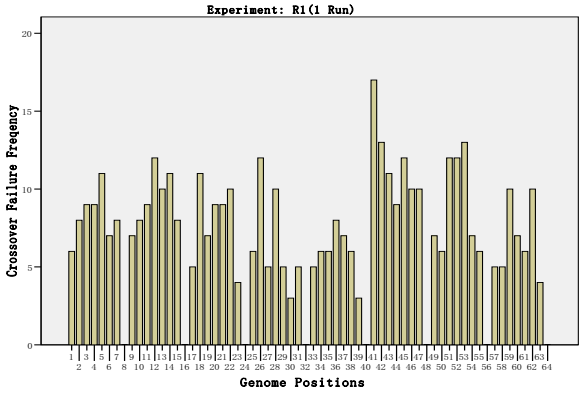

Figure 6: Crossover failures at each locus of genome of R1 in one run

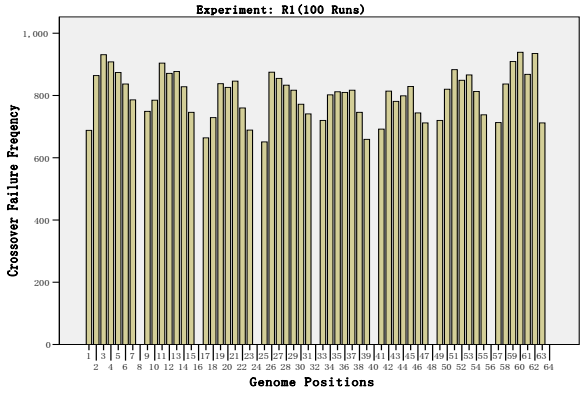

Figure 7: Accumulated crossover failures at each locus of genome of R1 over 100 runs

Figure 6 shows the BB structure of R1 clearly, revealing its eight building blocks. Those crossovers occurring on the BB boundaries yield no fitness loss, so the crossover failure tallies for these positions are "0", and indicate a BB boundary. In Figure 7, it is very easy to identify the BB structure. However, in each position of the same BB, the chance of a failed crossover is different. Most blocks display a rounded character, being highest in the center, showing that when crossover occurs in the middle part of a BB, there are more bits needing to be properly matched for the BB not to be a failure than when crossover occurs away from the middle.

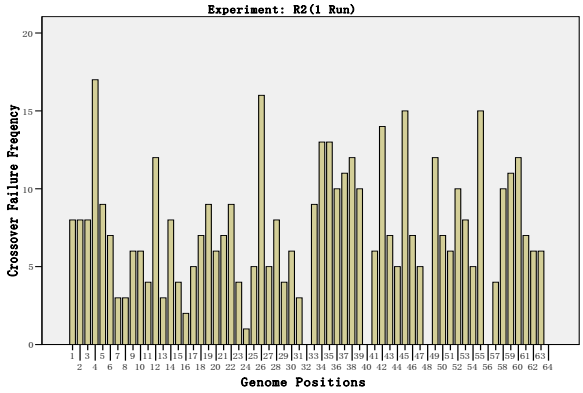

Figure 8: Crossover failures at each locus of genome of R2 in one run

R2's structure appears a bit vague from examination of a single run, but it still can be recognized as
(((B1,B2),B3,B4),B5,B6,B7,B8),
a weak hierarchical structure (Figure 8). In the aggregate plots of 100 runs (Figure 9), the structure is much clearer, of course, but it is only the single-run structure that is available to guide crossover during a run of an actual unknown problem. From Figure 9, it can be seen that the higher level the BB is on, the fewer crossover failures it generates. Position "32", the dividing point of two 32-bit combined BBs, generates no crossover failures. However, the next-lower-level BB boundaries, at positions "16" and "48", start to suffer failed crossvers, although their frequencies are still less than those of the next-lower-level BBs, whose boundaries are at positions "8", "24", "40" and "56".

RR(JH)'s 16 blocks are well separated by introns, where failures numbered either zero or one, as Figure 10 shows.

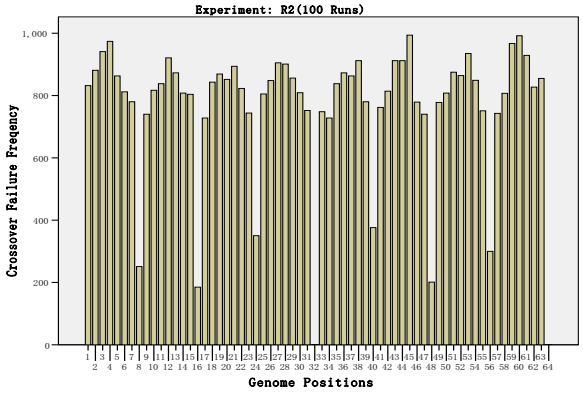

Figure 9: Accumulated crossover failures at each locus of genome of R2 over 100 runs

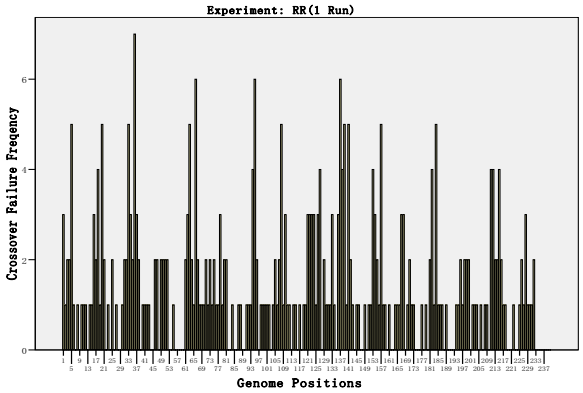

Figure 10: Crossover failures at each locus of genome of RR(JH) in one run

On the 100-run aggregate plots (Figure 11), the hierarchical structure is also very clear, although it is not evident in a single run. It is interesting that the BBs at each end suffer more failed crossovers than the BBs closer to the middle. This same phenomenon also appears in H-IFF, another hard function trapped by multi-level local optima. We guess that for those difficult problems needing a long evolution process, the BBs at each end are formed first, and therefore these earlier BBs will encounter more destructions than those BB that are formed later, in the middle. That lines up well with the reason that two-point crossover is generally better than one-point crossover, since two-point does better at preserving BBs at each end and exchanging non-BBs in the central part.

H-IFF's main hierarchical organization can be sketched from Figure 12. The highest-level BB is divided at position "32", and the next-highest BBs are found at positions "16" and "48", although the later is not as clear as the former. A much clearer picture of the BB structure can be seen in the summary of the 100 runs (Figure 13), but it is also evident that during the duration of a single run (Figure 12), enough information is obtained by tracking crossover statistics to allow the GA to capitalize on it during the course of the run.

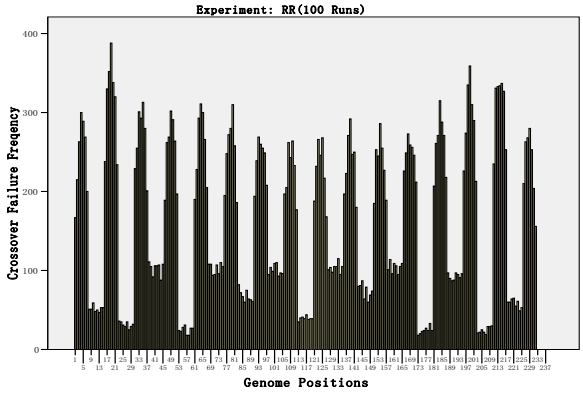

Figure 11: Accumulated crossover failures at each locus of genome of RR(JH) over 100 runs

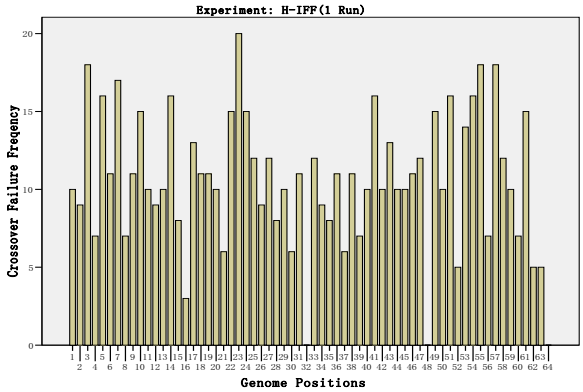

Figure 12: Crossover failures at each locus of genome of H-IFF in one run

5. DISCUSSION

5.1 Can Crossover Statistics Guide Mutation?

We initially supposed that the statistics collected from tracking crossover failures, from which the BB structure is learned, could also be useful in guiding mutation to focus only on BBs, thereby avoiding wasted effort on mutating introns. However, we find that there is a contradiction between the precondition of guided mutation and the crossover-failure statistics. For mutation to be usefully guided regarding a particular BB, it must be done before the BB is formed (i.e., mutation on an already discovered BB is normally harmful). But crossover failures are generated only after the BB is produced (and then broken), so the needed guidance is not available when the mutation operator could make use of it. That is to say, the guidance statistics, whose emergence indicates that there is already a BB existent near a given point, are not needed to guide mutation to evolve another BB at that point, since the BB is already discovered. However, for BBs in problems with multiple local optima, it may be useful to guide a non-single-bit mutation, to explore for other BBs as good as or better than the current BB.

6. CONCLUSIONS

The Building Block Hypothesis can be viewed as involving a hierarchical model in which BBs have different sizes

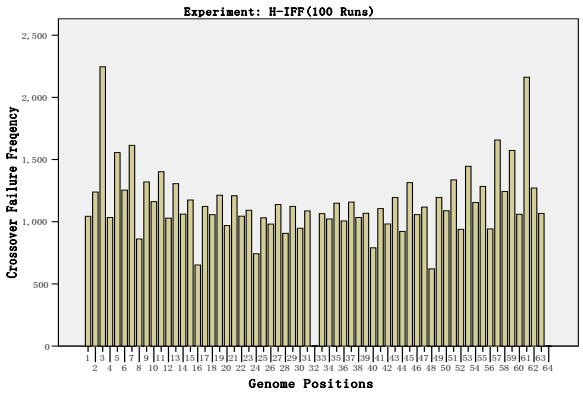

Figure 13: Accumulated crossover failures at each locus of genome of H-IFF over 100 runs

or granularities (or resolutions) at different levels. Various levels of BB discovery are characteristic of various stages in the evolution process.

Building Blocks (or linkages among nearby loci) can be observed by tracking failed crossovers, which are those that cause both offspring to be worse, or one to be worse and one unchanged, since such failures, absent more widely distributed epistatic interactions, assure that a building block on the crossover point is being disrupted.

Failed crossovers can therefore be used to bias the selection of crossover points to speed the GA's search, and simultaneously provide an outline of the genome structure, with negligible computational cost.

7. ACKNOWLEDGMENT

We wish to acknowledge the help of Prof. Lishan Kang, the support of the Michigan State University High Performance Computing Center, and the valuable suggestions of the reviewers.

8. REFERENCES

[1] C. Aporntewan and P. Chongstitvatana. A quantitative approach for validating the building-block hypothesis. In *2005 IEEE Congress on Evolutionary Computation*, pages 1403–1409. IEEE CEC 2005, September 2005.

[2] J. S. de Bonet, C. L. Isbell, Jr., and P. Viola. MIMIC: Finding optima by estimating probability densities. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 424–430. The MIT Press, 1997.

[3] S. Forrest and M. Mitchell. Relative building-block fitness and the building-block hypothesis. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109–126. Morgan Kaufmann, San Mateo, CA, 1993.

[4] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1989.

[5] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530, 1989.

[6] G. Harik. *Learning linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. PhD thesis, University of Michigan, 1997.

[7] G. Harik. Linkage learning via probabilistic modeling in the ECGA. Technical Report IlliGAL Report No. 99010, IlliGAL,University of Illinois at Urbana-Champaign, 1999.

[8] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. *IEEE-EC*, 3(4):287–297, November 1999.

[9] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[10] B. Howard and J. Sheppard. The royal road not taken: A re-examination of the reasons for GA failure on R1. In *Proceedings of the Genetic and Evolutionary Computation Conference(GECCO-2004)*, pages 1208–1219, June 2004.

[11] M. P. Lobo and D. E. Goldberg. A hierarchy machine: learning to optimize from nature and humans. *Complexity*, 8(5):36–45, 1999.

[12] R. S. Michalski. LEARNABLE EVOLUTION MODEL: Evolutionary processes guided by machine learning. *Machine Learning*, 38(1-2):9–40, 2000.

[13] M. Mitchell, S. Forrest, and J. H. Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. In F. J. Varela and P. Bourgine, editors, *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, 1991*, pages 245–254, Paris, 11–13 1992. A Bradford book, The MIT Press.

[14] H. MÜhelenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature (PPSN IV)*, pages 178–187, 1996.

[15] M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 511–518, San Francisco, California, USA, July 2001. Morgan Kaufmann.

[16] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532, Orlando, FL, July 1999. Morgan Kaufmann Publishers, San Fransisco, CA.

[17] J. D. Schaffer and A. Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 36–40. Lawrence Erlbaum Associates, Inc., July 1987.

[18] M. Sebag and M. Schoenauer. Controlling crossover through inductive learning. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 209–218, Berlin, 1994. Springer.

[19] D. Thierens and D. E. Goldberg. Mixing in genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 38–45. ICGA93', June 1993.

[20] R. A. Watson, G. S. Hornby, and J. B. Pollack. Modeling building-block interdependency. In *Proceedings of the Third Conference on Parallel Problem Solving from Nature(PPSN V)*, volume 1498, pages 97–106, 1998.

[21] R. A. Watson and J. B. Pollack. Hierarchically consistent test problems for genetic algorithms: Summary and additional results. In *Proceedings of the Genetic and Evolutionary Computation Conference(GECCO-1999)*, pages 292–297. Morgan Kaufmann, July 1999.