A Kernighan-Lin Local Improvement Heuristic that Softens Several Hard Problems in Genetic Algorithms

William A. Greene
Computer Science Department
University of New Orleans
New Orleans, LA 70148
bill@cs.uno.edu

Abstract: We present a local improvement heuristic for genetic algorithms, and term it a Kernighan-Lin style heuristic. We analyze the runtime cost of the heuristic, and show that it can be affordable. We then demonstrate through experiments that the heuristic provides very quick solutions to several problems which have been touted in the literature as especially hard ones for genetic algorithms. We suggest explanations for why the heuristic works so well on these problems.

1 INTRODUCTION

Genetic algorithms seek to find very good solutions to problems, by applying the forces of Darwinian evolution to a population of candidate solutions. The forces we have in mind are: survival of the fittest, mating with crossover, and mutation.

From now on, we will speak of a population of *individuals*, where an individual is a candidate solution. In this paper we will assume that an individual has a single chromosome, and in fact, is identified with that chromosome. Moreover, for us a chromosome will be a bit string, that is, a sequence of bits, where a bit can have value either 0 or 1. We assume all individuals have chromosomes which are the same length; throughout, we let N denote the number of bits in our individuals. Also we assume each individual can be evaluated for its fitness, where the fitness value is a non-negative real number.

Applying the forces of Darwinian evolution takes the following forms. Fitter individuals are given a higher probability of participating in mating, that is, survival of the fittest. In mating with crossover, the chromosomes of two parents are cut into fragments and those fragments then interchanged, to form the chromosome of each child. Mutation introduces some variation into the gene pool. For us, mutation will amount to flipping some bits.

With a little bit of luck during mating with crossover, occasionally a child will be formed that is fitter than its two parents. As the population evolves, we expect that fitter and fitter individuals will surface. Indeed, the Schema Theorem of John Holland (see [Holland, 1975] or [Goldberg, 1989]) shows that our expectation has a mathematical foundation.

As just noted, occasionally a child should appear that is fitter than its two parents. It is reasonable to ask, why rely only on crossover to produce individuals with enhanced fitness? A *local improvement heuristic* is one that makes one or several improvements to a child, just after its manufacture, and before entering it into the general population.

One local improvement heuristic is *hill-climbing*, which names a general heuristic search technique. For the individuals (chromosomes) that we are interested in, one step in hill-climbing an individual means: flip that bit which will result in the greatest improvement to fitness. As a local improvement operator for genetic algorithms, a child might undergo one step of hill-climbing, or several, or continue until no further improvement is possible. The latter choice, hill-climbing each child until no further improvement in the child is possible, has known problems. Children wind up in tight clusters around local maxima. Also, population diversity is lost, and there is no guarantee that the global optimum will be identified. Finally, children of local maxima may tend to be located in valleys.

As an extreme view, in this paragraph we argue that the global optimum can be found solely through local improvement, applied to any single individual. There is no need for mating with crossover, there is no need for an evolving population. Given a single individual, it differs from the (or an) optimal individual by having the wrong bit values on some subset of its bits. So, loop through all the possible subsets of bits. For each such subset, flip those bits, and record the fitness that results. When looping terminates, an individual of maximal fitness will have been found. Of course, this approach is infeasible. If individuals consist of N bits, there are 2^N subsets of bits, so this algorithm has exponential cost, which is intolerable.

2 A KERNIGHAN-LIN LOCAL IMPROVEMENT HEURISTIC

Our heuristic is inspired by a heuristic used in [Kernighan and Lin, 1970] for bisecting graphs. (Kernighan and Lin consider subsets of graph vertices which will change sides in the graph bisection at hand.) Thus we term our heuristic a Kernighan-Lin heuristic. Our local improvement heuristic for genetic algorithms falls in between the approaches of the two preceding paragraphs. Instead of considering all subsets of bits, it considers a heuristically chosen tower $S_1 \subseteq S_2 \subseteq S_3 \subseteq \ldots$

of subsets of bits. Subset S_k differs from S_{k-1} by including one more bit. A bit is chosen for addition to S_{k-1} in a manner similar to hill-climbing, namely, we choose a bit which is optimal with respect to change to fitness when flipped, but we allow that change to be a negative change (one no worse than the change from any other bit, we intend). We allow negative changes, in the hope that some later bit flip, in concert with previous flips, will result in a substantial improvement to fitness. In the metaphor of hill-climbing, we allow a descent into a valley, in the hope that we will reach a yet higher hill. Ultimately, we flip the bits in that S_k which gives the greatest increase in fitness. A similar idea, termed k-opt local search, and applied to the unconstrained binary quadratic programming problem, is found in [Merz and Freisleben, 2002].

In forming our tower $S_1 \subseteq S_2 \subseteq S_3 \subseteq \dots$, altogether how many bits should we be willing to flip? Half the bits, N/2, seems like a reasonable first guess, where, recall, N is the number of bits in an individual. We wrote our program in Java, which is object-oriented; consequently, we present our algorithm expressed in the style of object orientation. The algorithm is given in Figure 1.

2.1 COST ANALYSIS

The reader has no doubt already foreseen that our heuristic will figure fitness values many times, and so might conceivably incur prohibitive time costs. In this section we will compute the time cost of the heuristic. But first we make two observations. For a given problem, simply to calculate the change in fitness that results from flipping one bit may possibly be done at less cost than fully calculating the fitness of an arbitrary individual. With this in mind, in Figure 1 we have extracted this action as a separate subprogram, `newFitnessIfFlipBit(bit)`. Similarly, if an individual's fitness is already calculated, the cost of adjusting its fitness to account for a bit getting flipped may be less than the cost of a complete fitness calculation. Hence, the cost of adjustment appears in Figure 1 as the auxiliary subprogram `recalculateFitnessAfterFlipping(bit)`.

Let $f(N)$ be the runtime cost of our subprogram `newFitnessIfFlipBit(bit)`, as a function of N, where N is the number of bits in our individuals. Let $g(N)$ be the cost of `recalculateFitnessAfterFlipping(bit)`. Quite possibly $f(N)$ and $g(N)$ are the same cost, since the associated subprograms have similar responsibilities. We will use the standard "theta" notation for orders of magnitude of functions. Referring to Figure 1, step (1) takes time $\Theta(N)$. Since we are assuming that `maxFlips` equals N/2, the for-loop numbered as step (2) will have $\Theta(N)$ iterations. The inner for-loop at step (3) will have $\Theta(N)$ iterations, and its body, step (4), costs $\Theta(f(N))$. Thus altogether step (4) accumulates a cost that is $\Theta(N^2 f(N))$, and as we shall see, this is likely to be the dominating cost of the algorithm. Step (5) has cost $\Theta(N)$ each time it is per-

```
/* Make an improvement to this individual */
public void improveSelf()

     Wc := copy of this individual;     //working copy; undergoes bit flips
     WcList := empty list;              //stages of Wc as bits are flipped
(1)  availableBits := all the bits;     //bits available for flipping

(2)  for numFlips := 1 to maxFlips do{  //assume maxFlips = N/2
(3)     for each b in availableBits do
(4)        pair b with Wc.newFitnessIfFlipBit(b);
(5)     bestBit := that element of availableBits with the best paired value
                   (i.e., new fitness if that bit is flipped in Wc),
                   if there are several optimal bits, then toss a coin
(6)     Wc.flipBit(bestBit);                    //change Wc
(7)     Wc.recalculateFitnessAfterFlipping(bestBit);
(8)     WcList.appendAtEnd(copy of Wc);
(9)     remove bestBit from availableBits;
     }//end loop "for numFlips := 1 to maxFlips do"

(10) bestWc := that element of WcList with highest fitness;
     if this.fitness() < bestWc.fitness()      //if bestWc is an improvement
(11)    then change this individual to bestWc
}//end command improveSelf()


//auxiliary methods described:

/* Return the new fitness that would be obtained if the specified bit were
          to be flipped in this individual */
public float newFitnessIfFlipBit( bit );

/* Command to re-calculate fitness after a specified bit is flipped */
public void recalculateFitnessAfterFlipping( bit );
```

Figure 1: The Kernighan-Lin Local Improvement Heuristic
(Line numbers on the left are for discussion in the text.)

formed (actually we could incorporate this maximizing step within the preceding loop). Each performance of steps (6) and (8) should have constant cost. Step (7) has cost $\Theta(g(N))$ each time it is performed, so accumulates altogether a cost that is $\Theta(Ng(N))$. Step (9) has cost $\Theta(N)$. Thus the loop from step (2) through step (9) has a cost that is $\Theta(\max\{N^2 f(N), Ng(N)\})$. The remaining two steps must have lower cost than that, so our cost calculation is complete: the cost is $\Theta(\max\{N^2 f(N), Ng(N)\})$.

By way of example, if $f(N)$ and $g(N)$ were to have constant costs, which is not so inconceivable, then the cost of our local improvement heuristic is $\Theta(N^2)$. That cost may be quite acceptable, if it induces evolving populations to produce highly fit individuals very rapidly, that is, after just a few generations.

Also consider this case: suppose the fitness function is an algebraic expression of numerous subfunctions, each of which depends on a small subset of bits. Then $f(N)$ and $g(N)$ are likely far cheaper than a full fitness calculation.

3 THE GENETIC ALGORITHM

The genetic algorithm that we use holds no great surprises, but for the sake of completeness, we will briefly describe it. It is generational (versus steady state), that is, an (almost) entirely new population $P(t+1)$ for time $t+1$ is built from the preceding generation $P(t)$ at time t. In all experiments reported in this paper, population size is 40 individuals. Elitism is practiced: the fittest two elements of $P(t)$ automatically survive into generation $P(t+1)$. Then we fill up $P(t+1)$ to the same size as $P(t)$ by adding children which are produced from mating with crossover, using parents from $P(t)$. Individuals of $P(t)$ are selected for parenting by using a weighted roulette wheel, but first in a pre-step we scale population fitnesses to the real interval [1, 4]. Our crossover operator is 2-point crossover, which produces two children from two parents. (A detail: to increase population diversity, a child enters $P(t+1)$ only if an identical copy of it has not already been added earlier.) Once $P(t+1)$ has been filled, we apply mutation to its elements.

The only surprise we may have for the reader is that we practice fairly heavy mutation. Mutation is graduated and stochastic. Mutation is stochastic, in that mutation attempts are made, which do not necessarily result in an actual bit flip. And mutation is graduated in that less fit children in $P(t+1)$ are subjected to greater amounts of mutation. Specifically, the mutation phase begins by sorting $P(t+1)$ into decreasing fitness. Then the number of mutation attempts made on a child is proportional to its rank in the sorted population. There is a maximum number of mutation attempts; that maximum number is 40. Any mutation attempt succeeds to result in an actual bit flip only with a 50% chance. Thus, the child three-quarters of the way into sorted list $P(t+1)$ undergoes 3/4 * 40 = 30 mutation attempts, but we expect only about 15 of these to succeed to actual bit flips. (A detail: the two elite survivors enter $P(t+1)$ twice; the first copy is spared any mutation, the second copy undergoes mutation attempts as just described.)

There are several places where we apply our Kernighan-Lin local improvement heuristic. (1) The initial population $P(0)$ is first made up of random individuals, each of which then undergoes local improvement. (2) Each elite survivor undergoes local improvement (probably again) as it is being added to $P(t+1)$. (3) Each child undergoes local improvement, just after it is formed through crossover. (4) After being mutated, an individual undergoes local improvement.

In the problems we study, the maximal fitness is known. Consequently, we let generations $P(0)$, $P(1)$, $P(2)$, etc., evolve until an individual of maximal fitness surfaces or until some maximum number of generations has occurred.

4 FOUR HARD PROBLEMS FROM THE LITERATURE

In this section we experiment with four problems which have been touted in the literature as being ones especially difficult for genetic algorithms. Happenstantially, earlier researchers who worked on these problems asserted that niching was necessary for solving them. Our experiments will demonstrate, in particular, that our Kernighan-Lin local improvement heuristic can replace and indeed improve upon niching as the key for solving these problems.

4.1 THE ONE-DIMENSIONAL ISING PROBLEM

This problem is featured in [Van Hoyweghen, Goldberg, and Naudts, 2001]. Our definition of the fitness function is only superficially different from theirs. The N bits of an individual are treated as a ring, in that the first bit is considered to be the successor to the last one. The fitness function is defined as: add 1 each time a bit has the same value as its successor. Thus, maximum fitness is N and is achieved twice, by the string of all 0's and the string of all 1's. Minimum fitness is 0, and it is achieved by the two strings that alternate their bit values (when N is even). Using the terminology of the cited paper, a *domain* is a group of contiguous bits sharing the same value, and a *wall* is the interface where a 0 is succeeded by a 1, or the other way around. Let us add a term: a bit is *isolated* if its two neighbors have the opposing bit value. As isolated bit is a minimal domain, and both sides of it are walls.

The cited paper analyzes why this problem is difficult for a plain GA. Some examples of mutation will suggest the difficulty. Mutating an isolated bit causes fitness to increase by 2, since it eliminates 2 walls. On the other hand, mutating a bit in the middle of a non-trivial domain will cause fitness to decrease by 2, and mutating a bit at the wall where two non-trivial domains meet will not change the fitness value. Thus mutation does not seem to hold out great promise as an efficient mechanism for improving the population. The authors provide simple illustrations of how crossover between highly fit parents

can easily produce children of lower fitness. The authors assert that niching is a necessity for solving this problem.

We will show that our Kernighan-Lin local improvement heuristic can replace and indeed improve upon niching as the key to solving this problem. But first let us return to the cost issue of our heuristic. Its two supplemental algorithms, `newFitnessIfFlipBit(bit)` and `recalculateFitnessAfterFlipping(bit)`, each has constant cost for the problem at hand, since each only needs to examine the two neighbors of the flipping bit. The cost for our heuristic was computed in the general case in section 2.1; for the problem at hand its cost is $\Theta(N^2)$.

The results of our experiments with this problem are summarized in Table 1. There are 20 trials. On each trial, population size is 40, and a maximum of 500 generations is allowed. Each individual consists of N = 256 bits, so maximum fitness is 256. The first column gives the maximum number of bits which are allowed to flip on a local improvement operation. The second column gives the average final generation number, over the 20 trials. The third column gives the average time in seconds of a trial; where we were using a 1.7 GHz Pentium IV processor, and the source code was written in Java. Finally, the fourth column gives the average best fitness found, over the 20 trials.

The first detail line in Table 1 is the case that up to N/2 = 128 bits could be flipped on a local improvement operation. As the detail line shows, in an average of just 2.45 generations, with time cost 3.46 seconds per trial, an individual of maximum fitness 256 was found on every trial.

Let us compare this result with [Van Hoyweghen, Goldberg, and Naudts, 2001], who practice niching as sharing. For their best result (pp. 698-699 and Figure 3 of their paper), when individual size N = 80, and population size is 140, then on average their approach took 68 generations to find an individual of maximum fitness. We point out that our individuals are substantially longer (N = 256), our population substantially smaller (40), and the number of generations (2.45) very much smaller.

A good question is, how many bits should we be willing to potentially flip, on a local improvement operation? Earlier we said that a reasonable first guess is: half of them, N/2, which is quite a few. We wondered what would result if fewer than that many were flippable. The remaining detail lines of Table 1 show our results when the maximum number of flippable bits decreases to 64, 32, and finally 16. When we allow up to 64 bits to flip, the results are on a par with allowing 128 to flip. When we allow at most 32 bits to flip, it turns out that on 14 of the 20 trials an individual of maximum fitness was found in short order, namely, on generation 20.39 and time 11.83 seconds. A fifteenth trial found an optimal individual but only at generation 398. The remaining 5 trails exhausted the 500 generations, at which point their best individual had the best suboptimal fitness of 254, and consisted of two domains, one of 0's and the other of 1's, both domains being longer than the 32 allowable bits. The case of allowing only up to 16 bits to flip (the last detail line in Table 1) is worse yet.

Table 1: One-Dimensional Ising Problem

Number of trials = 20. Population size = 40. Maximum generations per trial = 500. Size of an individual = 256 bits. Maximal fitness = 256.

max flippable bits	avg. final gen. #	avg. trial time (sec.)	avg. best fitness
128	2.45	3.46	256
64	4.5	3.87	256
32	159.1	92.69	255.5
16	447.7	184.59	253.9

Why does the local improvement heuristic work so well on this problem? Here is our explanation. First to be flipped are the isolated bits, for they represent the best change to fitness, namely, an increase (by 2). Also, note such flips join the domains on either side of the isolated bit. Next the algorithm starts flipping bits at the walls; such flips leave fitness unchanged. Note that plain hill-climbing would stop after flipping just the isolated bits. Flipping a wall bit makes the domain on one side of it lengthen and makes the domain on the other side shorten. With some luck, domains of, say, 0's will lengthen to crowd out and eliminate all the remaining domains of 1's, and so form an individual of maximum fitness. Although the algorithm may get around to seeing what results when bits that are in the middle of domains get flipped (each time decreasing fitness by 2), the heuristic eventually declines to actually flip those (line 10 of Figure 1).

4.2 K-FOLD 3-DECEPTIVE FITNESS FUNCTION

This example is used in [Pelikan, Goldberg, and Sastry, 2001], as a test case for their Bayesian Optimization Algorithm (BOA). Let us assume that the number N of bits in an individual is a multiple of 3, and partition those bits into N/3 subgroups of 3 bits each. Each subgroup of 3 bits makes its own contribution to total fitness, but the contribution made by a subgroup of 3 bits is given by a certain deceptive function h. If u is the number of 1-bits in a subgroup, then

$$\begin{aligned} h(u) &= 1.0, \text{ if } u = 3 \\ &= 0.0, \text{ if } u = 2 \\ &= 0.8, \text{ if } u = 1 \\ &= 0.9, \text{ if } u = 0 \end{aligned}$$

The fitness of an individual is the sum of the fitnesses of the N/3 subgroups. There is one individual of maximum fitness N/3, that being the individual of all 1's. Every individual with the property that each of its subgroups consists of either three 1's or three 0's is a local optimum; there are $2^{N/3}$ such local optima.

Once again the two supplemental algorithms, `newFitnessIfFlipBit(bit)` and `recalculateFitness-AfterFlipping(bit)`, of our Kernighan-Lin

Table 2: k-Fold 3-Deceptive

Number of trials = 20. Population size = 40. Maximum generations per trial = 500. Size of an individual = 240 bits. Maximal fitness = 80.

max flippable bits	avg. final gen. #	avg. trial time (sec.)	avg. best fitness
120	1.0	2.32	80
60	3.15	2.70	80
30	21.5	11.13	80
15	46.15	18.41	80
6	76.5	21.80	80

local improvement heuristic have constant cost. The reason is that if we flip one bit, the change or adjustment to fitness only requires us to consider the subgroup of three bits in which the flipped bit lies. Therefore the heuristic itself has cost $\Theta(N^2)$.

The results of our experiments with this problem are given in Table 2. We let N = 240, so that maximum fitness is 240 / 3 = 80. Other experimentation parameters (number of trials, etc.) are the same as for the problem of section 4.1.

As the first detail line of Table 2 shows, when we allow up to N/2 = 120 bits to be flipped on each local improvement operation, then on every one of the 20 trials the genetic algorithm finds the individual of maximum fitness already on the first generation (after the randomized, then improved, initial population $P(0)$); moreover, it does this at an average time cost of just 2.32 seconds per trial. Other detail lines of Table 2 show results when we allow fewer and fewer bits to be flipped on a local improvement operation. Even when we allow as few as 6 bits to be flipped, still the genetic algorithm finds the individual of maximum fitness on every trial, and does so at not an extravagant time cost. The k-fold 3-deceptive problem ceases to be elusive in the face of our Kernighan-Lin local improvement heuristic. By way of comparison, [Pelikan, Goldberg, and Sastry, 2001] report (Figure 3, p. 525 of the cited paper) that, when individuals have a length of 150 bits, their BOA approach requires on average something like 160,000 fitness evaluations to unearth the individual of maximum fitness. It is not so easy to reckon the cost of our algorithm in terms of equivalent fitness evaluations, so we will let Table 2 speak for itself. That table shows that our Kernighan-Lin local improvement heuristic incurs a very low cost while it solves this problem on every trial.

4.3 HIERARCHICAL IF-AND-ONLY-IF (HIFF)

This very interesting test problem was introduced by [Watson, Hornby, and Pollack, 1998]. They sought a challenging problem, which a genetic algorithm, practicing niching, should be able to solve, if the building block hypothesis is true. They wanted their problem to be *decomposable*, that is, consist of subproblems, but not *separable* (subproblems should not have optima which sum to be the optimum for the whole problem). Subproblems should be *interdependent* (how one subproblem improves its fitness can depend on how others are doing so). Finally, the subproblems should comprise a *hierarchy*. The HIFF problem has found appeal with other researchers. For example, both the papers [Pelikan and Goldberg, 2000] and [Pelikan and Goldberg, 2001] interest themselves in this problem.

Here we give our own description of the HIFF problem. Imagine an individual which consists of 2^d bits. The intention is that fitness should consist of contributions made by a hierarchy of sets of partitioning subgroups of bits, where subgroups have size 2^c, for $0 \le c \le d$. It is helpful to think of the 2^d bits in our individual as the leaf nodes in a full binary tree. Thus, we will consider there to be 2^d - 1 auxiliary tree nodes on the scene, namely, the interior nodes in the tree, which we use to describe the fitness calculation. The root node of this tree is at depth 0, and the leaves (the 2^d bits in our individual) are at depth d. The interior tree nodes act as quasi-bits. Working upward in the tree from the leaves, assign a value of 0, 1, or "other" (we use 2 for the other value) to interior nodes p, as follows. Give p the value 0, if both children of p have value 0; give p the value 1, if both children of p have value 1; otherwise, give p the other value. Now fitness is calculated as the sum of the following contributions. The leaves all contribute 1 to fitness. Each interior node at depth c contributes 2^{d-c} if it has value 0 or 1, else it contributes nothing to fitness. Note that the root can potentially contribute 2^d; the two nodes below it can each potentially contribute 2^{d-1}, for a total of 2^d; the four grandchildren of the root can each potentially contribute 2^{d-2}, for a total of 2^d; and so on. The maximum fitness an individual can have is $(d+1)2^d$. It is achieved by two individuals, the one each of whose 2^d bits are 1's, and similarly for all 0's. (And minimum fitness is 2^d, since the leaves always contribute 1 each.)

Now we will calculate the cost of our Kernighan-Lin heuristic, for the HIFF problem. Recall that our notation is that N is the number of bits in an individual. Here $N = 2^d$. We claim that the subprograms `newFitnessIfFlipBit(bit)` and `recalculateFitnessAfterFlipping(bit)` each have cost proportional to d, that is, log(N). We reason as follows. As regards the calculation of fitness, each of the 2^d bits of an individual is naturally paired with one of its two neighbors. If the bit has a different value than the neighbor, then its ancestors in the full binary tree all have the "other" value. Flipping the bit will cause its tree parent to change to value 0 or 1 and make a contribution to fitness, and potentially the same is true of its tree grandparent, and so on up the tree, until possibly an ancestor is encountered that must retain its "other" value. On the other hand, if a bit has the same value as the neighbor, flipping it will necessitate changing its tree parent (and potentially tree grandparent, and so on) to the "other"

Table 3: Hierarchical If and Only If (HIFF)

Number of trials = 20. Population size = 40. Maximum generations per trial = 500. Size of an individual = 256 bits. Maximal fitness = 2304.

max flippable bits	avg. final gen. #	avg. trial time (sec.)	avg. best fitness
128	1.85	3.69	2304
64	3.05	3.55	2304
32	5.8	4.14	2304
16	28.5	14.96	2304
8	376.85	149.53	2137.6

value and subtracting the contribution(s) formerly made. Thus, what is needed by both the Kernighan-Lin subprograms cited earlier is a traversal up the tree, with cost $\Theta(d)$. The cost of the Kernighan-Lin algorithm was calculated in general in section 2.1. For the HIFF problem, the cost becomes $\Theta(N^2\log(N))$.

Will this be a tolerable cost? The experiments which we next report show that it is indeed tolerable, and it immensely reduces the number of generations which must evolve before finding an individual of maximal fitness. We let each individual consist of $2^8 = 256$ bits, and then maximum fitness is 9*256 = 2304. Other experimentation parameters (number of trials, etc.) are the same as for experiments with our earlier problems. Our results are summarized in Table 3.

The first detail line is the case that up to N/2 = 128 bits could be flipped on a local improvement operation. As the detail line shows, in an average of just 1.85 generations, with time cost 3.69 seconds per trial, an individual of maximal fitness 2304 was found on every trial. The remaining detail lines of Table 2 show our results when the maximum number of flippable bits decreases to 64, 32, 16, and finally 8. All but the last case show excellent performance, with an individual of maximal fitness being found on every trial, in surprisingly few generations and with very reasonable time cost. For the last case, when at most 8 bits could be flipped on a local improvement operation, it turns out that for 7 of the 20 trials an individual of maximal fitness was unearthed, and on the remaining 13 trials, one of the two individuals (first half segment of the bits are one of the values 0 or 1, and the second half segment are the other value) of next best fitness 2048 is the best individual found by the limit of 500 generations.

We now compare our results with [Watson, Hornby, and Pollack, 1998] who devised the HIFF Problem. For them, individuals consist of 64 bits. They try several variants of genetic algorithms on this problem, with best results coming when 2-point crossover is used, and population diversity is enforced by niching through fitness-sharing. In their Figure 2, we read that using a population size of 1,000, an individual of maximum fitness was consistently found on 10 trials, apparently requiring no more than 400 generations. We note that our own individuals are substantially bigger (256 bits *vs.* 64), our populations much smaller (40 *vs.* 1,000), and the number of generations required very much smaller (2 or 3, say, *vs.* 400).

What makes our local improvement heuristic work so well on the HIFF problem? Our belief is that the local improvement heuristic abets two phenomena, namely, (1) homogeneous chunks get formed, and then (b) the majority bit value (0 or 1) supplants the other one. As for (a), consider the following. Imagine a block of 2^c bits, which are all 1's except for one 0. Consider the interior tree node v that is the nearest ancestor for the entire block. None of the ancestors of that errant 0 bit, up through v, make a contribution to fitness (nor beyond v either), but flipping the 0 bit makes those ancestors suddenly contribute, that is, makes fitness jump higher. Thus, the errant 0 bit is likely to get flipped early on by the heuristic. Next, if the block has exactly two errant 0's, then flipping either will result in a higher fitness, so the two are likely to be chosen for flipping. Moreover, flipping one of them will increase the benefit (yet higher fitness contribution) of flipping the other. Or, we can look at phenomenon (a) from the other direction. Once a block of 2^c bits has become homogeneous, the heuristic will take a long time to choose to flip a bit within it, since to do so causes a number of contributions to fall away.

As for phenomenon (b), consider the following. Suppose the first half of an individual's bits are 0's the and second half are 1's. Flipping any bit then has the same effect on fitness, namely, fitness decreases by $X = 2 + 2^2 + 2^3 + ... + 2^{d-1}$ since each non-root ancestor of the flipped bit no longer contributes to fitness (already beforehand the root was not contributing). Suppose the algorithm tosses a coin and happens to choose to flip a 0 bit. Thereafter, to flip a 1 bit will decrease fitness by X, whereas to flip a 0 bit either decreases fitness by less than X or even increases fitness. Thus the remaining 0 bits will be chosen for flipping, but none of the 1 bits.

4.4 HIERARCHICAL TRAP FUNCTIONS 1 & 2

The fitness function for the HIFF problem has an interesting hierarchical structure, but there is no deceptiveness to it on any level of the tree of interpretations. The next two problems introduce such deceptiveness, using deceptive functions similar to the one for the k-fold 3-deceptive fitness function. These two problems are taken from [Pelikan and Goldberg, 2001], who are the problem devisers. The authors use these two problems as test cases for a hierarchical version of the Bayesian Optimization Algorithm (BOA). They add niching to the BOA in order to make it hierarchical (that is, able to succeed on fitness functions with a hierarchical structure). The authors assert that a genetic algorithm requires effective linkage learning, plus niching, if it is to reach the global optimum in these two problems. We will demonstrate, as earlier, that our Kernighan-Lin local improvement heuristic can replace niching. Together with the linkage learning that

occurs in 2-point crossover, our heuristic will identify the global optimum very quickly.

4.4.1 H-Trap-1

For this problem an individual consists of 3^d bits. As in the HIFF problem, it is helpful to think of the 3^d bits in our individual as the leaf nodes in a full tertiary tree. There are $(3^d - 1) / 2$ interior nodes in the tree, that is, above the 3^d bits comprising the individual *per se*. Each interior node is given an interpretation value, which is either 0 or 1 or an "other" value. Also, each interior node makes a contribution to fitness, which is given by a function of the values (0 or 1 or "other") of its three children.

The interpretation value assigned to an interior node is defined to be: 0, if its three children have interpretation value 0; but 1, if its three children have interpretation value 1; or the "other" value otherwise. The latter option includes the cases that some child has interpretation value "other", or that the interpretation values of the three children are, say, two 0's together with one 1.

Next we say what fitness contribution is made by the tree nodes. As for the 3^d nodes of the individual *per se*, they make no contribution to fitness (this is unlike the HIFF problem). The contribution to fitness made by an interior node depends on the depth of the node, and also upon the interpretation values of its three children. Suppose the node is at depth c (the root is at depth 0; at depth d-1 are the parents of the 3^d bits of the individual *per se*). If any child has an interpretation value that is the "other" value, then the parent makes no contribution to fitness; otherwise, the contribution made by a parent, other than the tree root, is given by $H(u) * 3^{d-c}$; here u is the number of child interpretation values which are 1's, and $H(u)$ is the trap function defined by

$$\begin{aligned} H(u) &= 1.0, \quad \text{if } u = 3 \\ &= 0.0, \quad \text{if } u = 2 \\ &= 0.5, \quad \text{if } u = 1 \\ &= 1.0, \quad \text{if } u = 0 \end{aligned}$$

As for the contribution made by the root, the above plan is followed except that we replace function H by another one, H_r , which is deceptive and defined as

$$\begin{aligned} H_r(u) &= 1.0, \quad \text{if } u = 3 \\ &= 0.0, \quad \text{if } u = 2 \\ &= 0.5, \quad \text{if } u = 1 \\ &= 0.9, \quad \text{if } u = 0 \end{aligned}$$

Note that every level of interior nodes has the potential to contribute altogether 3^d to fitness. Maximum fitness is therefore $d * 3^d$, and is achieved by one individual, all of whose 3^d bits are 1's. Further note that at each level in the interpretation tree below the root, although function H does equally favor either three 0's or three 1's for children, it also tilts toward favoring three 0's. Finally, at the root, function H_r makes the fitness function in fact favor 1's everywhere.

Table 4: Hierarchical Trap Function No. 1

Number of trials = 20. Population size = 40. Maximum generations per trial = 500. Size of an individual = 243 bits. Maximal fitness = 1215.

max flippable bits	avg. final gen. #	avg. trial time (sec.)	avg. best fitness
121	1.0	3.94	1215.0
81	1.0	2.74	1215.0
27	3.6	3.45	1215.0
9	13.55	6.29	1215.0
3	484.5	153.17	832.3

The results of our experiments on H-Trap-1 are given in Table 4. For this problem we let an individual consist of $N = 3^5 = 243$ bits, that is, depth $d = 5$. Maximum fitness is therefore $5 * 3^5 = 1215$. Other experimentation parameters (number of trials, etc.) are the same as for the previous experiments. The first detail line is for the case that we allow one-half the bits to flip on any local improvement operation. Then for subsequent detail lines we allow up to one-third, one-ninth, etc., of the bits to flip. The first three detail lines display approximately the same time cost, and find the individual of maximum fitness on every one of the 20 trials. The fourth detail line shows the time cost increasing, but the heuristic is still serving to unearth the optimal individual on all 20 trials. Only the last detail line shows the collapse of the effectiveness of the local improvement heuristic.

4.4.2 H-Trap-2

The second hierarchical trap function H-Trap-2 is very similar to the first. In fact, the only difference will be the contribution made at non-root interior nodes of the interpretation tree which sits atop the 3^d bits that comprise the individual *per se*. The earlier function H did tilt in the direction of favoring three 0's for the child values but in fact made the same fitness contribution when the child values were three 1's as when they were three 0's. Now function H is to be replaced by a new function H' which, at non-root nodes, makes a greater fitness contribution when the three child values are 0's. Again we will let an individual consist of $N = 3^5 = 243$ bits. We define function H' by

$$\begin{aligned} H'(u) &= 1.0, \quad \text{if } u = 3 \\ &= 0.0, \quad \text{if } u = 2 \\ &= 0.5, \quad \text{if } u = 1 \\ &= 1.02, \quad \text{if } u = 0 \end{aligned}$$

Note that H' agrees with H except that $H'(0)$ is slightly greater than $H(0)$. (Our definition of the second hierarchical trap function H-Trap-2 is only superficially different from that in [Pelikan and Goldberg, 2001].) We continue to use function H_r as the means for making a fitness contribution at the root itself. It then turns out that the individual of maximum fitness is again the one all of

whose 3^d bits are 1's, and maximum fitness is again $d * 3^d = 1215$.

Our experimental results for H-Trap-2 are summarized in Table 5. Although problem H-Trap-2 appears a notch harder than H-Trap-1, the detail lines of Table 5 show that performance on it is rather similar to that for H-Trap-1.

By way of comparison, [Pelikan and Goldberg, 2001] apply their hierarchical BOA to these two hierarchical trap problems, and report that that algorithm has similar performance on the two problems. For H-Trap-1 we are informed (their Figure 2(c), page 517) that, in the case that individuals consist of 243 bits, on average 225,000 fitness evaluations are needed by their hierarchical BOA to find the individual of maximum fitness. Those researchers do not report a time cost for their experiments. Tables 4 and 5 show that our Kernighan-Lin local improvement heuristic has a rather low time cost, which may well surprise the reader.

Table 5: Hierarchical Trap Function No. 2

Number of trials = 20. Population size = 40. Maximum generations per trial = 500. Size of an individual = 243 bits. Maximal fitness = 1215.

max flippable bits	avg. final gen. #	avg. trial time (sec.)	avg. best fitness
121	1.0	5.47	1215.0
81	1.0	2.75	1215.0
27	3.2	3.03	1215.0
9	12.45	5.77	1215.0
3	448.0	142.78	893.9

5 CONCLUSION

We have presented a Kernighan-Lin style local improvement heuristic for genetic algorithms. We have analyzed the runtime expense of the heuristic, and shown that it is potentially affordable. We have utilized the heuristic on five problems from the literature, which earlier researchers have touted as very challenging ones for genetic algorithms, ones they say require niching for success. Our experiments show that our heuristic can not only replace niching as a key to success, but also improve upon it. Global optima are found after only a handful of generations, with time costs that are very reasonable.

When applied to the three hierarchical problems studied in this paper, in general the heuristic appears to create beneficial chunks for the hierarchy, at least within enough members of the population, that the beneficial chunks are soon assembled into globally optimal individuals, under the effects of the heuristic, in concert with the evolutionary forces at work in a genetic algorithm.

Its performance on the problems studied in this research suggests that the heuristic can potentially be beneficial for many problems, especially hierarchical ones. The heuristic appears to quickly form building blocks within individuals in the population, and to quickly steer from local optima to global optima.

REFERENCES

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley Publishing.

Holland, John (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

Kernighan, B., and S. Lin (1970). "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Systems Technical Journal*, vol. 49, pp. 291-307.

Merz, P., and B. Freisleben (2002). "Greedy and Local Search Heuristics for Unconstrained Binary Quadratic Programming". *Journal of Heuristics*, vol. 8, pp. 197-213.

Pelikan, M., and D. E. Goldberg (2000). "Hierarchical Problem Solving and the Bayesian Optimization Algorithm", in D. Whitley *et al*. (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO 2000), pp. 267-274. Morgan Kaufmann Publishers, San Francisco.

Pelikan, M., and D. E. Goldberg (2001). "Escaping Hierarchical Traps with Competent Genetic Algorithms", in L. Spector *et al*. (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO 2001), pp. 511-518. Morgan Kaufmann Publishers, San Francisco.

Pelikan, M., D. E. Goldberg, and K. Sastry (2001). "Bayesian Optimization Algorithm, Decision Graphs, and Occam's Razor", in L. Spector *et al*. (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO 2001), pp. 519-526. Morgan Kaufmann Publishers, San Francisco.

Van Hoyweghen, C., Goldberg, D. E., and Naudts, B. (2001). "Building Block Superiority, Multimodality, and Synchronization Problems", in L. Spector *et al*. (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO 2001), pp. 694-701. Morgan Kaufmann Publishers, San Francisco.

Van Hoyweghen, C., Goldberg, D. E., and Naudts, B. (2002). "From TwoMax to the Ising model: easy and hard symmetrical problems", in W. B. Langdon *et al*. (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO 2002), pp. 626-633. Morgan Kaufmann Publishers, San Francisco.

Van Hoyweghen, C., Naudts, B, and Goldberg, D. E. (2002). "Spin-Flip Symmetry and Synchronization." *Evolutionary Computation*, vol. 10(4), pp. 317-344.

Watson, R. A., Hornby, G. S., and Pollack, J. B. (1998). "Modeling Building Block Interdependency," in *Parallel Problem Solving from Nature 1998* (PPSN V), pp.97-106. Springer-Verlag, Berlin.