Gray-Box Optimization using the Parameter-less Population Pyramid

Brian W. Goldman
BEACON Center for the Study of Evolution in Action
Michigan State University, U.S.A.
brianwgoldman@acm.org

William F. Punch
BEACON Center for the Study of Evolution in Action
Michigan State University, U.S.A.
punch@msu.edu

ABSTRACT

Unlike black-box optimization problems, gray-box optimization problems have known, limited, non-linear relationships between variables. Though more restrictive, gray-box problems include many real-world applications in network security, computational biology, VLSI design, and statistical physics. Leveraging these restrictions, the Hamming-Ball Hill Climber (HBHC) can efficiently find high quality local optima. We show how 1) a simple memetic algorithm in conjunction with HBHC can find *global* optima for some gray-box problems and 2) a gray-box version of the Parameter-less Population Pyramid (P3), utilizing both the HBHC and the known information about variable relationships, outperforms all of the examined algorithms. While HBHC's inclusion into P3 adds a parameter, we show experimentally it can be fixed to 1 without adversely effecting search.

We provide experimental evidence on NKq-Landscapes and Ising Spin Glasses that Gray-Box P3 is effective at finding the global optima even for problems with thousands of variables. This capability is complemented by its efficiency, with running time and memory usage decreased by up to a linear factor from Black-Box P3. On NKq this results in a 375x speedup for problems with at least 1,000 variables.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

Keywords

Local Search; Global Search; Parameter-less

GECCO '15, July 11 - 16, 2015, Madrid, Spain
© 2015 ACM. ISBN 978-1-4503-3472-3/15/07...$15.00
DOI: http://dx.doi.org/10.1145/2739480.2754775

1. INTRODUCTION

Recent work [10, 2] has shown that providing local search with more problem specific information than just an evaluation function can greatly improve search power and efficiency. These improvements require information easily derived for a broad range of interesting NP-Hard problems.

Here we extend those benefits to global optimization on binary problems by integrating this local search with additional heuristics [1]. We introduce a novel algorithm designed to combine local and global search as simply as possible. To test the true power of this process we also extend the Parameter-less Population Pyramid (P3) [4], the current state-of-the-art in black-box optimization [5]. This "Gray-Box P3" utilizes both efficient local search and the additional problem information to drastically improve performance and reduce memory usage.

2. GRAY-BOX DOMAIN

In black-box optimization, the only information available is a specification for what constitutes a valid solution and a function which returns the quality of a given solution. While these very simple requirements allow almost any optimization task to fit into this domain, many problems have additional knowledge which could be used by search algorithms. At the other end of the spectrum is white-box optimization, in which everything about the problem is known. Search techniques in this domain specialize at the cost of generality. Gray-box optimization exists between these two extremes, exposing additional information which is available for a variety of problems to be exploited by search.

Here we shall consider the gray-box optimization domain with the following characteristics [2]: The function which evaluates the quality of a solution is determined by the summation of subfunctions. Each subfunction uses at most k variables in the given solution, such that k remains constant as the number of problem variables N increases. Each subfunction is independently evaluable and the variables each subfunction uses are known.

While more restrictive than black-box optimization, this domain is still very general. MAX-SAT, and by extension all NP-hard problems, can be represented as a summation of subfunctions which evaluate each clause [10]. Previous approximation studies of Vertex Cover [6], Set Cover [12], MAX-CUT [3], and Ising Spin Glasses [7] all work with problems which fit in the gray-box domain. These relate back to problems of real world interest such as network security, computational biology, VLSI design, and statistical physics.

In order to obtain efficiency bounds for optimization algorithms, the following assertions are also included [2]: The total number of subfunctions grows no more than linearly with N, which holds true for randomized MAX-SAT, mesh spin glasses, and NK Landscapes. Furthermore, each variable participates in at most c subfunctions, which holds true for mesh spin glasses and Nearest Neighbor NK. If violated, these assertions do not necessarily prevent the gray-box optimization algorithms from solving the problem, but may result in slower than predicted performance.

3. HAMMING-BALL HILL CLIMBING

The first algorithm to rigorously exploit the features of the gray-box domain described in Section 2 was [10]. In this study a hill climber was developed that found the approximate next-best move in $\mathcal{O}(1)$ time given $\mathcal{O}(N)$ setup. This is achieved by first determining the fitness effect of making each move, and then updating only effected moves each time a change to the solution is made. Due to the requirements of the domain, at most $\mathcal{O}(ck)$ moves can be effected, with each requiring at most $\mathcal{O}(1)$ subfunction calls to update their fitness effects. Combined with move binning, this technique is able to perform both first improvement and next-best hill climbing in $\mathcal{O}(I + N)$, where I is the number of improving moves required to reach a local optimum. In comparison, the black-box approach for these searches can require $\mathcal{O}(IN^2)$.

By again leveraging the properties of this domain, the Hamming-Ball Hill Climber (HBHC) [2] extends this method to find solutions that cannot be improved by flipping r or fewer bits called r-bit local optima. While naïvely this would require testing $\mathcal{O}(N^r)$ moves, in the gray-box domain only $\mathcal{O}((3ck)^r N)$ must be checked. This is due to the bounded relationships between variables. If i and j do not share a subfunction, the effect of flipping both variables must be equal to the sum of flipping both separately. Therefore, only variables that share a subfunction must be flipped together when checking $r = 2$. The problem of determining which variables must be flipped together is equivalent to finding the connected induced subgraphs of G with r or fewer vertices, such that each vertex in G is a variable and an edge exists between two vertices if and only if those variables share a subfunction. In [2] an upper bound is derived for how many moves this could create, but no algorithm was given to efficiently discover these moves. This process was not included in their timing calculations as it only has to be performed once per problem instance.

Under the assertion that $r \ll N$ and that r does not increase with N, the Hamming-Ball Hill Climber requires only $\mathcal{O}(I + N)$ time to find r-bit local optima starting from a random solution. As I is likely in $\mathcal{O}(N)$ this means finding random r-bit local optima is no more than a constant amount slower than generating random solutions.

While these techniques make it possible to choose the best move just as efficiently as choosing any random improving move, the findings of [10] suggest that, especially when paired with subsequent search heuristics, using random improving moves is more effective. In [2] this decision was amended slightly to include the requirement that smaller distance moves are chosen before larger distance moves.

3.1 Novel r-order Subgraphs Algorithm

In order to determine which moves in the hamming-ball must be evaluated, we developed ConnectedInducedSubgraphs given in Figure 1. CISG is a recursive helper function which finds all subgraphs which contain a given *subset* and a given vertex v, while excluding any other vertices added to *closed*. To find all subgraphs, CISG is called once for each vertex in the graph, such that *closed* contains all previously searched vertices, and *subset* = *open* = $\emptyset$. In the initial call all desired subgraphs which contain v are found, which is why v remains in *closed* to prevent duplicate subgraphs from being returned.

```
 1: procedure ConnectedInducedSubgraphs
 2:     $closed \leftarrow \emptyset$
 3:     $found \leftarrow [\,]$
 4:     for all $v \in V$ do
 5:         $closed \leftarrow closed \cup \{v\}$
 6:         $found \leftarrow found + \mathrm{CISG}(v, \emptyset, closed, \emptyset)$
 7:     return $found$
 8: procedure CISG($v$, $subset$, $closed$, $open$)
 9:     $subset' \leftarrow subset \cup \{v\}$
10:     $found \leftarrow [subset']$
11:     if $|subset'| \geq r$ then return $found$
12:     $closed\_here \leftarrow \emptyset$
13:     $open' \leftarrow open \cup adjacent(v)$
14:     for all $v' \in open'$ such that $v' \notin closed$ do
15:         $closed\_here \leftarrow closed\_here \cup \{v'\}$
16:         $closed \leftarrow closed \cup \{v'\}$
17:         $recurse \leftarrow \mathrm{CISG}(v', subset', closed, open')$
18:         $found \leftarrow found + recurse$
19:     $closed \leftarrow closed - closed\_here$
20:     return $found$
```

Figure 1: Algorithm to recursively find all connected induced subgraphs of size r or fewer.

At each recursive level CISG expands *open* to include any vertices adjacent to v in the graph. By construction this means that *open*$'$ contains all possible ways of adding a single vertex to the current *subset*$'$. As each v' is tested it is temporarily added to *closed* to prevent recursive calls from adding it again to *subset*$'$.

When applied to the sparse graphs inherent in the gray-box domain, this algorithm requires $\mathcal{O}(r!(ck)^r N)$ time, which reduces to $\mathcal{O}(N)$. The time spent in each call is dominated by the loop over *open*$'$. In the worst case, *open*$'$ increases in size by the full adjacency of v, which is bounded by ck. This creates a worse case complexity for a single top level call of $\prod_i^r ick = r!(ck)^r$. This must be called once for each of the N variables resulting in $\mathcal{O}(r!(ck)^r N)$.

4. TOURNAMENT UNIFORM CROSSOVER: TUX

Hamming-Ball Hill Climbing is not sufficient to efficiently find the global optima on problems with even moderate epistasis [2]. This is because, like all random restart hill climbers, it relies on random initialization to fall inside the global optima's basin of attraction.

To remedy this limitation, we set out to develop a minimally complex memetic algorithm to help increase this probability. Figure 2 presents the Tournament Uniform Crossover (TUX) algorithm, which combines simplistic selection with equal probability uniform crossover, the most basic unbiased

```
procedure ITERATE-TUX
    Create random $solution$
    Hamming-Ball Hill Climb $solution$
    for all $T_i \in T$ do
        if $T_i$ is empty then
            $T_i \leftarrow solution$
            return
        Cross $solution$ with $T_i$ to create $2^{i+1}$ offspring
        Hamming-Ball Hill Climb each offspring
        $solution \leftarrow$ best of offspring, $solution$, and $T_i$
        $T_i \leftarrow$ empty
    Add $solution$ to end of $T$
```

Figure 2: One iteration of TUX optimization. T is an ordered list of solutions, each position of which could be empty, awaiting a crossover partner.

```
procedure ITERATE-P3
    Create random $solution$
    Apply hill climber
    if $solution \notin hashset$ then
        Add $solution$ to $P_0$
        Add $solution$ to $hashset$
    for all $P_i \in pyramid$ do
        Mix $solution$ with $P_i$
        if $solution$'s fitness has improved then
            if $solution \notin hashset$ then
                Add $solution$ to $P_{i+1}$
                Add $solution$ to $hashset$
```

Figure 3: One iteration of P3 optimization. *pyramid* is an ordered set of populations and *hashset* is a set of all solutions in *pyramid*.

crossover, to generate starting solutions likely to be in the global optima's basin of attraction.

Conceptually TUX iteratively builds a structure similar to a single elimination bracket for solutions. Each "match" in the tournament takes in two candidate solutions, produces offspring via uniform crossover, applies hill climbing to each offspring, with the "winner" being the best of all those solutions. The tournament "bracket" is constructed iteratively, storing an initially empty list of solutions T, such that $|T|$ is equal to the height of the tournament. Iterative construction is possible because when a sub-bracket is complete only a single solution emerges and solutions only need to be stored until their partner is found. TUX is fully elitist but does not prematurely converge. This is because search continuously integrates new randomly generated solutions through other parts of the bracket. Whenever the top of the current bracket is reached, TUX doubles the size of the virtual tournament.

When crossing solutions at T_i, TUX produces 2^{i+1} offspring. This relationship ensures that in total all levels of the tournament, including random initialization, perform the same number of hill climbing steps. It also shifts the focus of search toward areas expected to be of higher fitness. The expectation is also that it becomes progressively harder to improve solutions the higher up the tournament you advance, so more attempts are necessary to create new useful solutions.

The primary advantages of TUX are that it does not introduce any new parameters (though it still requires an r for the hill climber) and is relatively simple to implement. Even so it allows for learning from previous local optima and as Section 6 will show it is quite effective at optimization.

5. PARAMETER-LESS POPULATION PYRAMID: P3

5.1 Origins

The Parameter-less Population Pyramid (P3) is a recently introduced black-box optimization method which requires no problem specific configuration [4, 5]. Unlike previous parameter-less methods it was shown to be more efficient than competing state-of-the-art algorithms across a number of problem classes.

P3 combines hill climbing and model building using a novel, iteratively constructed, pyramid of populations. Figure 3 provides the high level description of how the algorithm works. To exploit information from multiple solutions, P3 performs global optimal mixing [9]. Each time a solution is added to a level of the pyramid P_i, linkage learning is performed to create a collection of variable clusters. These clusters, collectively referred to as a linkage tree, are learned from the pairwise variable entropy of P_i. When a solution is mixed with P_i each cluster is used to donate variable values from a random donor in P_i into the solution. After each donation the modified solution is evaluated, with the change being reverted if solution quality decreased. In this way P3 performs multiple evaluations per mixing event. This also means that unlike TUX, which uses information from only two solutions to perform crossover, P3 uses information from an entire population. Due to this model building, P3 requires $\mathcal{O}(N)$ amortized time per evaluation and $\mathcal{O}(N^2)$ memory usage.

5.2 Gray-Box Specialization

P3 represents a natural method for integrating the HBHC into a global optimization algorithm as P3 already utilizes local search. While HBHC's inclusion adds a parameter, Section 6.1 and Section 6.2 provide evidence that r can be fixed to 1, preserving the parameter-less nature of P3. Beyond using HBHC, there are also a number of ways in which P3 can be made more efficient by leveraging the additional information available in the gray-box domain.

First and foremost, linkage learning is no longer necessary. By definition the direct non-linear relationships between variables are known. As a result P3 no longer needs to store the pairwise frequency information which leads the black-box version to require $\mathcal{O}(N^2)$ memory. Instead, we have developed a method for creating a linkage tree which learns clusters from the same dependency graph defined in Section 3. The goal is for each cluster to be a connected induced subgraph, with the size of clusters mirroring those produced by the agglomerative linkage learning process normally used with P3. To form a single cluster, a random graph search is performed from a random starting vertex until a desired number of unique vertices have been explored. Cluster sizes are set recursively. For each cluster of size $l > 1$ a cluster of size a and a cluster of size $l - a$ are also created, with a chosen uniformly from the range $[1..l-1]$. This recursive process begins by initializing $l = N$ and splitting to form the first two clusters.

This linking algorithm has a number of useful properties. First, it creates exactly $2N - 2$ clusters, distributed in size similarly to the black-box clustering algorithm. Performing a random graph search to find l unique vertices requires $\mathcal{O}(lck)$ time, meaning cluster creation is optimally efficient. The cluster splitting process has identical properties as random pivot quicksort, meaning the sum of cluster sizes is $\mathcal{O}(N \log N)$ in the average case. This efficiency allows new clusters to be created before every mixing event, unlike Black-Box P3 where they are created only when new solutions are added to the population. For simplicity the clusters are shuffled after they are created, not sorted on size like in Black-Box P3.

Beyond efficiency, there are good reasons to believe these clusters will be useful for search. The closer two variables are in the dependency graph, the more likely they are to appear in the same cluster. All variables on average are expected to appear in at least one cluster, but variables which are central in the graph will appear in more clusters than those on the periphery. The more paths of a given length between two variables, the higher the probability of them being in the same cluster. Unlike Black-Box P3, this linkage tree does not require clusters to be nested, allowing more diversity in the types of clusters appearing in a single tree.

In effect the clusters are sampling moves which the HBHC would make if $r \geq l$. For any solution in the search space which is not globally optimal, there must exist some move which will improve its quality. However, this move may be arbitrarily large and it is intractable to test all possible moves of even moderate size. By sampling from all possible large moves, we can maintain tractability while gaining a potentially non-zero probability of improvement. As clusters are used to donate values between solutions, these moves are always in the direction of previously found high quality solutions. This assumes that the density of high quality solutions is higher than average between good solutions.

Another efficiency gain possible due to the gray-box domain is that each time a donation is made, only the effected part of the solution's fitness needs to be recalculated. As a result the number of subfunction evaluations required to determine the change in fitness is only $\mathcal{O}(l)$. This also allows for Gray-Box P3 to efficiently reapply hill climbing after each donation as only effected moves need to be rechecked. As a result, a donation and its resulting modifications from hill climbing are kept only if the new local optima is at least as fit as the solution before the donation occurred. All combined a single donation plus returning to a local optimum requires $\mathcal{O}(l + I)$ time, while just the donation in Black-Box P3 requires $\mathcal{O}(N)$.

6. EXPERIMENTS

To compare these gray-box optimization techniques we have chosen NKq-Landscapes [2] and Ising Spin Glasses [8]. NKq-Landscapes create a collection of randomly generated problem instances given a higher level problem class description. Each instance is described by a series of N subfunctions, each corresponding to a variable in the solution. This subfunction uses its variable and K other variables in the solution to calculate a fitness value. Fitness values are represented as a randomly generated lookup table, such that table entries are integers in the range $[0..q-1]$. As each subfunction reads $K+1$ variables, the table's size and q are set to 2^{K+1}. The quality of a solution is equal to the sum of the values returned by these subfunctions.

In this work we shall consider two methods for choosing the K variables each subfunction depends on: Nearest Neighbor NKq and Unrestricted NKq. In Nearest Neighbor NKq each variable depends on the K variables which sequentially follow it in the solution, with dependencies wrapping around the end of the solution. Landscapes of this form can be solved in polynomial time [11], allowing comparisons of how quickly each optimization algorithm can find the global optima. Nearest Neighbor NKq also ensures that $c = k = K + 1$ and that both c and k do not increase as N increases, meaning the efficiency conclusions made in Section 3 are applicable.

Unrestricted NKq landscapes draw the K dependencies at random without replacement. For $K > 1$ it is NP-Hard to find the global optimum of these landscapes. This also means that while k remains fixed, the maximum number of subfunctions a variable appears in (c) can increase as N increases. As a result some of the efficiency claims in Section 3 may not be applicable.

Ising Spin Glasses are a type of MAX-CUT problem relevant in statistical physics. Each spin glass encodes spins (vertices) and their relationships (edges) with the goal of assigning each spin a direction which minimizes relationship energy. Just as with NKq-Landscapes, Ising Spin Glasses as a whole are NP-Hard, but the $2D \pm J$ subset is polynomially solvable [8][1]. In this subset the graph is a 2D toroidal grid, with edge weights of ± 1. In gray-box terms, problems of this subset have $k = 2$ and $c = 4$ regardless of N.

For each problem class tested, we generated 50 instances. Extreme problem sizes were chosen to stress each algorithm. Each method was run once on each instance, and limited to 3 hours of computation and 4 GB of memory. Runs were performed on 2.5GHz Intel Xeon E5-2670v2 processors using the C++11 code available from our website.[2] Each time the run achieved a new best fitness we recorded the current amount of processing time used. Timing includes the discovery of subgraphs to allow for comparison between different radius values. When reporting the "best" fitness for an instance we mean the best fitness found by any method before the time limit is reached. On all Nearest Neighbor NKq instances the "best" fitness is also the global optimum, verified using dynamic programming. For Ising Spin Glass the "best" fitness was the global optimum 44 out of 50 times.

All figures report the median, upper and lower quartiles for either percentage error or seconds to reach the best. A run's percentage error is equal to the difference between its fitness and the best, divided by the best. When reporting seconds to reach the best fitness, any run which did not find the best fitness is treated as slower than any run that did. If the median run was unsuccessful, no data point is drawn.

6.1 The Effect of Radius

The only algorithm parameter in the HBHC, TUX, and Gray-Box P3 is the radius of the hamming-ball. Therefore our first experiments are designed to determine the effect of this parameter on solution quality.

Figure 4 shows the effect on final solution fitness as r increases. As expected from [2] the HBHC obtains higher qual-

[1]http://www.informatik.uni-koeln.de/spinglass/
[2]https://github.com/brianwgoldman/GrayBoxOptimization/releases

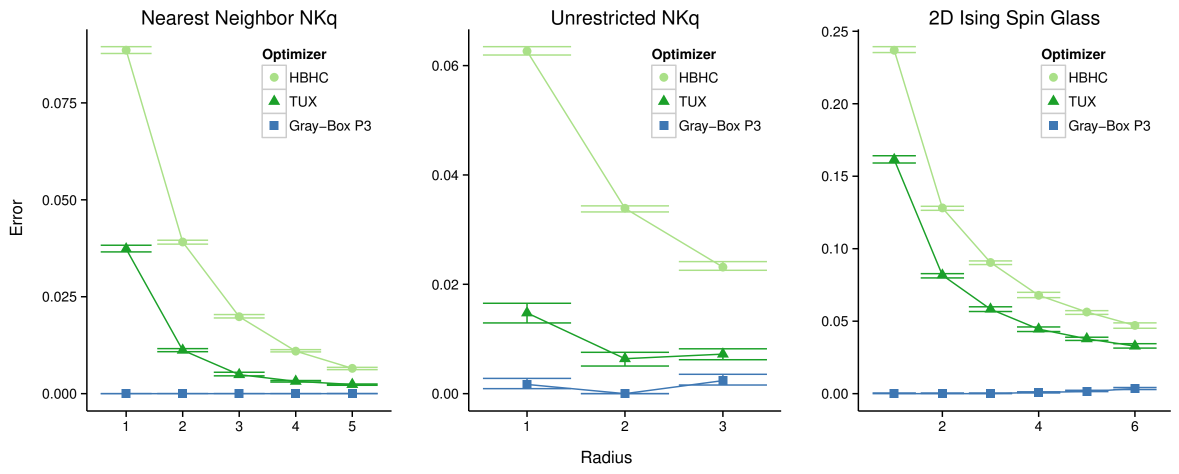

Figure 4: Comparison of how radius effects solution quality at termination. For NKq-Landscapes $N = 6,000$ and $K = 4$ and for Ising Spin Glasses $N = 6,084$. Range of radius values limited by memory constraints.

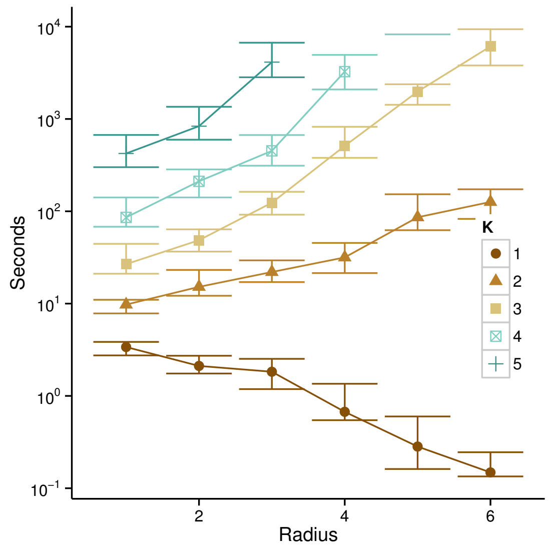

Figure 5: Time required for Gray-Box P3 to reach the global optimum of Nearest Neighbor NKq instances with $N = 6,000$.

ity as r increases, with the magnitude of the improvement decreasing. TUX has a similar relationship and outperforms HBHC on all three problems for all r values. Regardless of r, Gray-Box P3 outperforms both, with almost all r values reaching the same best fitness. For Nearest Neighbor NKq, Gray-Box P3 finds the global optimum in every run for $r < 4$, with only a single unsuccessful run at $r = 4$.

To further examine the effect of r on Gray-Box-P3, Figure 5 shows the number of seconds required to reach the global optimum. Setting $r = 1$ was the most efficient configuration for all $K > 1$, supporting the trend that Gray-Box P3 works best with small r values. With $K = 1$, the landscape is smooth enough that with a sufficiently high r the HBHC is able to find the global optimum.

6.2 Fitness Over Time

Reaching high quality solutions quickly can sometimes be more important than reaching the global optimum eventually. Figure 6 shows how solution quality progresses during search for each algorithm. HBHC and TUX have significant early delays caused by their high r values. Larger r's require a large amount of initial partial evaluation before performing hill climbing. After one full iteration HBHC effectively stalls, with TUX continuing to improve. Both are eclipsed by Gray-Box P3, which quickly descends to the global optimum, outperforming HBHC and TUX at every time point.

Figure 7 further illustrates the effect of r on Gray-Box P3. On Nearest Neighbor NKq and Ising Spin Glasses, increasing r does not change the shape of the curve. Instead the quality reached is simply time shifted, such that given more time higher r values will reach the same quality. As a result, for these problems we conclude that higher r values simply add more expense for no overall gain. On Unrestricted NKq this relationship is less certain, with $r = 1$ potentially having a different, and worse, shape than $r > 1$. However, due to memory and time restrictions it is difficult to know if this trend continues.

6.3 Scalability

Perhaps the most critical test of an optimization algorithm's quality is how it scales as problem difficulty increases. To test this behavior, we ran all three algorithms using the optimal r values determined experimentally in Section 6.1, varying N from 200 to 10,000 for NKq and 196 to 6,084 for Ising Spin Glass. In these plots we also include the black-box version of P3 to show the efficiency gains available for using gray-box information.

Figure 8 and Figure 9 show how long each algorithm required to reach the best overall quality found on Nearest Neighbor NKq and Ising Spin Glasses, respectively. For Nearest Neighbor NKq the best found is the global optimum for all runs of all lengths, while for Ising Spin Glasses the best quality found by any method was worse than the global optimum in 7 runs of $N = 4,096$ and 21 runs of $N = 6,084$. The median run of the HBHC was unable to

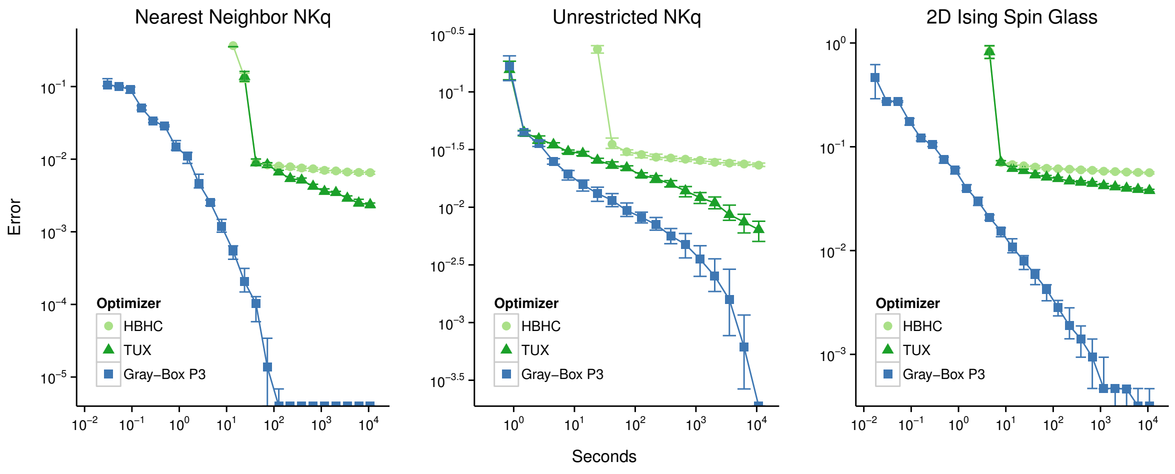

Figure 6: Comparison of solution quality during optimization on a log-log scale for different algorithms. For NKq-Landscapes $N = 6,000$ and $K = 4$ and for Ising Spin Glasses $N = 6,084$. Each algorithm uses its best found r value.

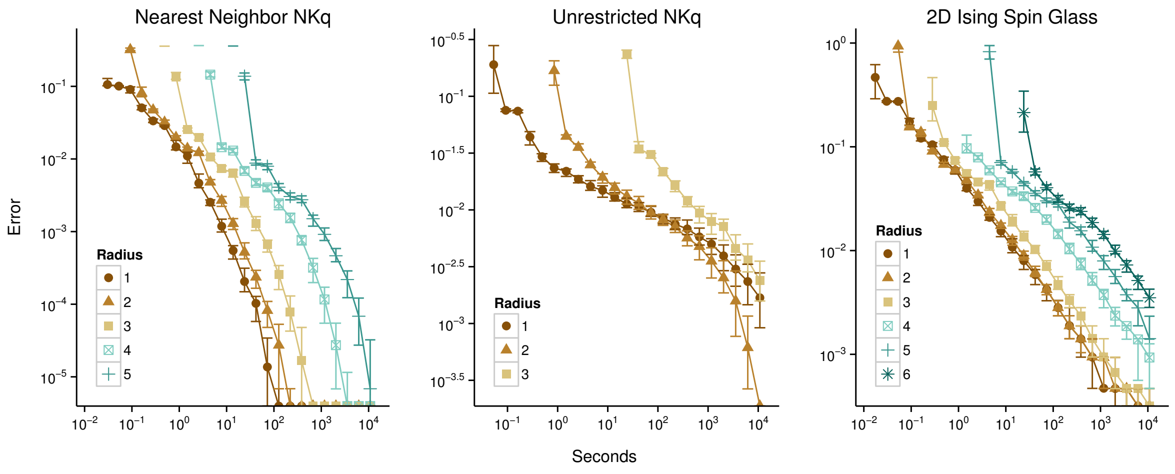

Figure 7: Comparison of Gray-Box P3's solution quality during optimization on a log-log scale for different r values. For NKq-Landscapes $N = 6,000$ and $K = 4$ and for Ising Spin Glasses $N = 6,084$.

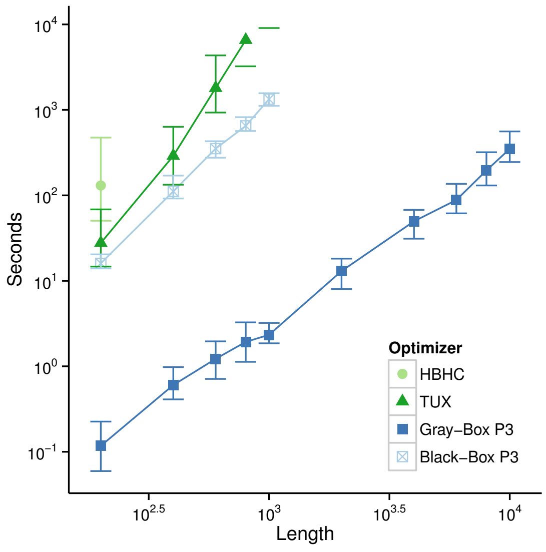

Figure 8: Comparison of how each algorithm's time required to reach the best fitness found scales with problem size on Nearest Neighbor NKq with $K = 4$. With $N = 1000$ Gray-Box P3 is 375x faster than Black-Box P3.

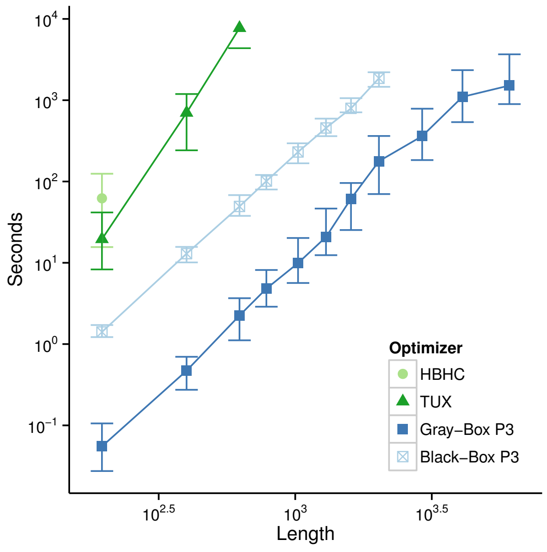

Figure 9: Comparison of how each algorithm's time required to find the best fitness found scales with problem size on Ising Spin Glass. With $N = 2025$ Gray-Box P3 is 4.6x faster than Black-Box P3.

reach the best fitness for any problems tested using more than 200 bits. TUX performed somewhat better, reaching the best fitness more than half of the time on problem sizes up to $N = 800$ and $N = 625$ for Nearest Neighbor NKq and Ising Spin Glass, respectively. Black-Box P3, which does not utilize partial reevaluation or the HBHC, was able to consistently reach the best fitness until it it hit the memory limit on $N = 2,000$ for NKq and $N = 2,916$ for Ising Spin Glass. This limitation is due to Black-Box P3's $\mathcal{O}(N^2)$ memory requirements.

For all sizes of both problems, Gray-Box P3 was the fastest to reach the best fitness. On Nearest Neighbor NKq the improvement is very significant, with no alternative finishing within two orders of magnitude. On its largest successful problem size, the mean time to completion for Black-Box P3 was 375 times slower than Gray-Box P3. Using the Mann-Whitney test to compare their run times results in $p < 10^{-16}$. This is especially impressive considering previous work has shown Black-Box P3 is faster to reach the global optimum than other leading black-box methods [5]. Applying regression, we estimate that Black-Box P3's time to global optimum on Nearest Neighbor NKq is $\mathcal{O}(N^{2.75})$ while Gray-Box P3's is $\mathcal{O}(N^{1.98})$.

The results on Ising Spin Glass are similar, with a less extreme difference between Black-Box P3 and Gray-Box P3. In general Gray-Box is the fastest technique to find the global optimum by an order of magnitude, with Black-Box P3's mean run finishing 4.6 times slower than Gray-Box on $N = 2,025$. Using the Mann-Whitney test to compare their run times results in $p < 10^{-14}$. The regression line suggests that while Gray-Box P3 scales at $\mathcal{O}(N^{3.35})$, Black-Box P3 scales at $\mathcal{O}(N^{3.05})$.

As Unrestricted NKq does not have a known global optimum, and the different algorithms rarely found the same best fitness, Figure 10 compares the error for each technique at termination. Gray-Box P3 in general finds the best fitness, with TUX occasionally performing better. When $N > 2,000$, Gray-Box P3 finds better quality solutions that all other methods for every instance. Using the Mann-Whitney test to compare the fitness of Gray-Box P3 and TUX when $N = 10,000$ results in $p < 10^{-16}$. HBHC and Black-Box P3 only reach similar qualities as TUX and Gray-Box P3 when $N = 200$, doing so in 4 and 7 runs, respectively. As the problem size increase TUX begins to fall behind Gray-Box P3, with the HBHC stabilizing at about 2.5% worse than the best found. Here Black-Box P3 performs worse than the other techniques, falling further behind as the problem size increases.

7. DISCUSSION AND CONCLUSIONS

In line with previous work, we have found that HBHC cannot effectively find global optima on problems with even moderate epistasis. In general it also obtains almost no improvement in fitness after only a few restarts. We designed TUX as a simplistic way of choosing restart locations based on previously found local optima. Unlike HBHC alone, TUX was able to continue improving given more time, finding global optima on problems three times as large.

TUX's effectiveness is likely due to the HBHC acting as a super repair operator for uniform crossover. Given a sufficiently large r the HBHC can return sections of the crossover offspring to either parents' original version of a given subfunction. The HBHC is elitist meaning there is a bias toward returning to the better of the two parent's versions. Furthermore, by being so disruptive, uniform crossover potentially allows for the HBHC to also find unrelated improvements.

While TUX improves over plain HBHC, Gray-Box P3 is required to perform truly successful global optimization. Gray-Box P3 replaces naïve local search with the HBHC and utilizes known non-linear relationships instead of statis-

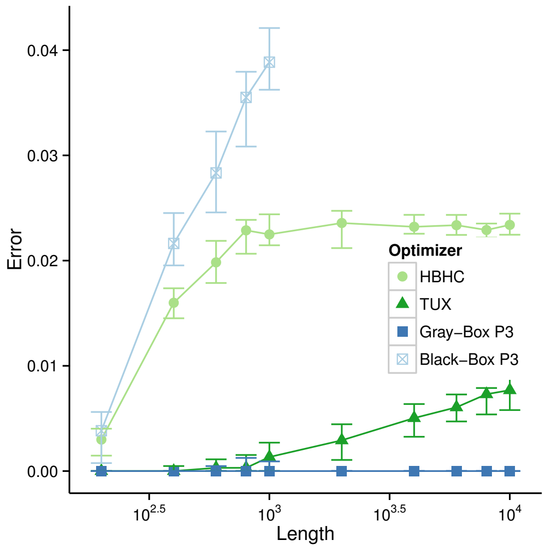

Figure 10: Relative qualities of each method as problem size increases on Unrestricted NKq with $K = 4$.

tical linkage learning. On NKq-Landscapes this drastically improves search effectiveness. A major source of this improvement is likely how difficult it is for Black-Box P3 to learn linkage relationships on these landscapes. Furthermore, Gray-Box P3 can perform partial reevaluation and efficient hill climbing during the mixing phase.

Gray-Box P3's success is less dramatic on Ising Spin Glasses. While it still outperformed all competitors, Black-Box P3 may actually scale better to larger problems. One explanation for this deviation is that Ising Spin Glasses require more exploration of equal fitness plateaus. For instance in Figure 6 and Figure 7 there is a significant pause in improvement when Gray-Box P3 reaches the second best fitness in the landscape. Nothing in its design suggests that Gray-Box P3 should be more effective at neutral drift. Another potential issue is that on these landscapes the importance of each non-linear relationship may be detectably unequal. As a result Black-Box linkage learning may better cluster variables which having meaningful impact on fitness while Gray-Box assumes all are equally important. A useful direction for future work would be to explore methods of performing efficient learning on top of the known variable interactions.

Somewhat surprising is the difference in behavior between the polynomially solvable problems and Unrestricted NKq. While Black-Box P3 performed very well in the former, it was the least successful in the latter. The optimal radius for Gray-Box P3 also shifted from 1 to 2. A potential explanation is that in both Ising Spin Glass and Nearest Neighbor NKq the number of unique variables r or fewer steps away from a given variable is significantly lower than the worst case. This explains why on Unrestricted NKq even moderately high r values hit our memory limit. For Black-Box P3 this may also be causing increased difficulty in linkage learning as variables become indirectly dependent on much larger sets. Furthermore, Black-Box P3 may be benefiting from an increased rate of duplicate dependencies on Nearest Neighbor NKq not present in Unrestricted NKq.

While the inclusion of HBHC into Gray-Box P3 introduces a parameter, it requires trivial configuration. In the worst case there may be a handful of r values to test. Furthermore, our evidence suggests setting $r = 1$ is quite powerful, with higher values likely to be only a time shift in quality. This is in contrast to r's role in HBHC, where low r values are never expected to reach the same quality has higher r values. Therefore we conclude that Gray-Box P3 maintains the out-of-the-box quality of Black-Box P3, while drastically improving efficiency for this new domain of problems.

8. REFERENCES

[1] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan. A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*, 15(5):591–607, 2011.

[2] F. Chicano, D. Whitley, and A. M. Sutton. Efficient identification of improving moves in a ball for pseudo-boolean problems. In *GECCO '14: Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 437–444, Vancouver, BC, Canada, 12-16 July 2014. ACM.

[3] P. Festa, P. M. Pardalos, M. G. Resende, and C. C. Ribeiro. Randomized heuristics for the max-cut problem. *Optimization methods and software*, 17(6):1033–1058, 2002.

[4] B. W. Goldman and W. F. Punch. Parameter-less population pyramid. In *GECCO '14: Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 785–792, Vancouver, BC, Canada, 12-16 July 2014. ACM.

[5] B. W. Goldman and W. F. Punch. Fast and efficient black box optimization using the parameter-less population pyramid. *Evolutionary computation*, 2015.

[6] P. S. Oliveto, J. He, and X. Yao. Analysis of the (1+1)-ea for finding approximate solutions to vertex cover problems. *Evolutionary Computation, IEEE Transactions on*, 13(5):1006–1029, 2009.

[7] M. Pelikan and D. E. Goldberg. Hierarchical BOA solves ising spin glasses and MAXSAT. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1271–1282, Chicago, 12-16 July 2003. Springer-Verlag.

[8] L. Saul and M. Kardar. The 2d±j ising spin glass: exact partition functions in polynomial time. *Nuclear Physics B*, 432(3):641–667, 1994.

[9] D. Thierens and P. A. N. Bosman. Optimal mixing evolutionary algorithms. In *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 617–624, Dublin, Ireland, 12-16 July 2011. ACM.

[10] D. Whitley, A. E. Howe, and D. Hains. Greedy or not? best improving versus first improving stochastic local search for maxsat. In *AAAI*, 2013.

[11] A. H. Wright, R. K. Thompson, and J. Zhang. The computational complexity of N-K fitness functions. *IEEE Transactions on Evolutionary Computation*, 4(4):373–379, Nov. 2000.

[12] Y. Yu, X. Yao, and Z.-H. Zhou. On the approximation ability of evolutionary optimization with application to minimum set cover. *Artificial Intelligence*, 180–181:20–33, 2010.