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

Real world applications of evolutionary techniques are often hindered by the need to determine problem specific parameter settings. While some previous methods have reduced or removed the need for parameter tuning, many do so by trading efficiency for general applicability. The Parameter-less Population Pyramid (P3) is an evolutionary technique that requires **no parameters** and is still broadly effective. P3 strikes a balance between continuous integration of diversity and exploitative elitist operators, allowing it to solve easy problems quickly and hard problems eventually. When compared with three optimally tuned, state of the art optimization techniques, P3 always finds the optimum at least a constant factor ***faster*** across four benchmarks (Deceptive Trap, Deceptive Step Trap, HIFF, Rastrigin). More importantly, on three randomized benchmarks (NK Landscapes, Ising Spin Glasses, MAX-SAT), P3 has a lower order of computational complexity as measured by evaluations. We also provide outlines for expected runtime analysis of P3, setting the stage for future theory based conclusions. Based on over 1 trillion evaluations, our results suggest P3 has wide applicability to a broad class of problems.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance, Experimentation

Keywords

Linkage Learning; Local Search; Parameter-less

1. INTRODUCTION

Providing a good solution in reasonable time to a broad class of real world black-box problems is a fundamental goal of evolutionary computation. Achieving this goal requires an optimizer to make assumptions about the nature of the problem it is solving as no optimizer can solve all problems equally well [15]. Many genetic algorithms (GAs) introduce evolutionary parameters which provide user control of search assumptions. These parameters can have enormous impact on search efficiency [3] and can have non-obvious optimal settings which require time-consuming tuning to discover [5]. When applied to black-box problems with non-trivial evaluation functions, the tuning process can be intractable, forcing users to rely on stock values or attempt a small number of test values.

The problem of parameter tuning has encouraged work to remove parameters from user control in an effort to improve out-of-the-box GA quality. In an early example, a crossover-based GA was devised which used concurrent racing populations which removed all user specified parameters [7]. Population racing has also been paired with more advanced evolutionary system, such as Hierarchical Bayesian Optimization [10]. Without the need to set a population size, racing ensures the GA will take no more than a logarithmic factor increase in evaluations beyond optimal. While experimental evidence suggests racing is only a constant factor loss, this method explicitly trades efficiency for applicability.

In this work we shall introduce the Parameter-less Population Pyramid (P3). P3 is an iterative method for constructing a collection of populations that requires no user parameters. It leverages entropy based linkage detection to learn how to efficiently mix solutions without problem specific information beyond the evaluation function. Perhaps most importantly, unlike other parameter-less techniques it experimentally appears to be at least a constant factor ***improvement*** over comparable, optimally configured, optimization methods.

2. THE P3 ALGORITHM

Unlike many GA techniques, P3 does not have a single population of solutions. Instead P3 maintains a pyramid-like structure, such that each level of the pyramid has a population of solutions, with each level representing something akin to different generations of evolution. Figure 1 provides a high level summary of how P3 iteratively constructs these populations and utilizes them to improve randomly generated solutions through hill climbing and crossover. While P3 can be applied to any discrete value representation, we will focus on the binary domain here for simplicity.

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2576768.2598350.

```
procedure ITERATE-P3
    Create random solution
    Apply hill climber (Section 2.1)
    if solution ∉ hashset then
        Add solution to P_0 (Section 2.2)
        Add solution to hashset
    for all P_i ∈ pyramid do
        Mix solution with P_i (Section 2.3)
        if solution's fitness has improved then
            if solution ∉ hashset then
                Add solution to P_{i+1}
                Add solution to hashset
```

Figure 1: One iteration of P3 optimization. *pyramid* **is an ordered set of populations and** *hashset* **is a set of all solutions in** *pyramid*.

```
procedure CLUSTER-CREATION
    unmerged ← {{0}, {1}, {2}, ..., {N − 1}}
    useful ← unmerged
    while |unmerged| > 1 do
        C_i, C_j ← min_{C_i,C_j ∈ unmerged} D(C_i, C_j)
        unmerged ← unmerged − {C_i, C_j} + {C_i ∪ C_j}
        useful ← useful + {C_i ∪ C_j}
        if D(C_i, C_j) = 0 then
            useful ← useful − {C_i, C_j}
    Order useful based on cluster size, smallest first.
    Remove largest cluster from useful.
    return useful
```

Figure 2: Algorithm describing how clusters are created using Equation 1 for each P_i. *unmerged* **and** *useful* **are ordered sets of sets of gene loci.**

2.1 Hill Climber

We use the First Improvement Hill Climber (FIHC),[1] which acts as follows. Consider all gene loci x_i in a random order. Evaluate the solution using all possible values of x_i. If any alternative value is a strict fitness improvement over the original value for x_i, replace x_i with the improved value. Iteration of the process continues until no such change results in a strict fitness improvement.

To avoid wasting evaluations, FIHC keeps track of all loci tested since the last change was made to the solution. If the algorithm returns to the same loci without any changes, it moves on to the next loci without performing any evaluations.

FIHC solves all linear functions in $O(N)$ time, where N is the number of genes. This is in contrast to methods like Steepest Ascent Hill Climber [1, 4] which require $O(N^2)$ as changes are only accepted after all neighbor solutions are evaluated. In general, FIHC is designed to find optima using a minimum number of evaluations. Furthermore, because FIHC tests loci in a random order it does not cause any positional bias to the search.

2.2 Pyramid

As mentioned, P3 maintains a pyramid like structure of populations with more optimized solutions in higher levels. More specifically, a solution's level is one higher than its highest parent, similar to the layering done in [8].

The levels represent disjoint sets of unique solutions, such that $\forall_{i,j \in P, i \neq j} P_i \cap P_j = \emptyset$. This relationship is maintained constructively, such that new solutions can only be added to a level if they do not exist in any level of the pyramid. This uniqueness check can be achieved in constant time by maintaining a hash of all solutions in the pyramid.

Along with storing the solutions, each level also maintains a table of pairwise gene value frequencies for use by crossover. This is also built constructively, such that initially all frequencies are set to 0. Each time a solution is added to a level, all pairs of loci are queried for their gene values. The table is then updated, incrementing the count for each observed pairing. This requires $O(N^2)$ operations (no evaluations) each time a solution is added to a level of the pyramid, but is constant in the number of solutions currently in the level.

[1]We use the name FIHC for descriptive clarity. This algorithm likely already exists in the literature under various alternative names.

2.3 Crossover

P3 uses a crossover method derived from that defined for LTGA [14] (Section 3.2); P3 uses gene value entropy to construct clusters of genes that should have their values conserved during crossover. There are two steps in defining this type of crossover: *cluster creation* and *cluster usage*.

During cluster creation, P3 creates a binary tree of clusters such that the leaves represent individual gene loci and each internal node represent the subset of loci created by joining the subsets encoded in its children. Using the table of observed frequencies described in Section 2.2, Equation 1 calculates the linkage "distance" between clusters of loci using the average pairwise entropy, as given in Equation 2.

$$D(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{c_i \in C_i} \sum_{c_j \in C_j} 2 - \frac{H(C_i) + H(C_j)}{H(C_i \cup C_j)} \quad (1)$$

$$H(C) = -\sum_{s \in S} p_c(s) \log(p_c(s)) \quad (2)$$

Figure 2 depicts the process of agglomeratively constructing the linkage tree. Starting with each gene location in separate clusters, Equation 2 is used to find which C_i and C_j in *unmerged*, the set of gene clusters, have the tightest linkage (minimum distance). These are merged to form a new cluster $C_i \cup C_j$, and the separate pieces are removed from *unmerged* while *useful* tracks all clusters ever created in the hierarchy. This process iterates until all locations have been merged into a single cluster. With proper bookkeeping this algorithm will construct the linkage tree from the frequency table in $O(N^2)$ [6].

Not all of the resulting clusters in the linkage tree are useful for crossover. The most obvious useless cluster is one that contains all the loci. A crossover that preserves all loci in a solution cannot create a new individual. We further remove clusters that are completely contained within their direct superset, such that if $D(C_i, C_j) = 0$, C_i and C_j are not used when performing crossover. $D(C_i, C_j)$ only returns 0 when the loci in C_i perfectly predict the gene values in C_j and vice versa. From this we can conclude that the relative settings for those loci should never be disrupted. When used on particularly small populations this has the potential to greatly reduce the number of clusters used during crossover.

```
procedure CLUSTER-USAGE
    for all C_i ∈ useful do
        for all d ∈ shuffled(P_i) do
            Copy d's gene values for C_i into solution
            if solution was changed then
                if solution's fitness decreased then
                    Revert changes
                break
```

Figure 3: Algorithm describing how clusters are used to perform crossover.

For instance, if $|P_i| = 1$, no clusters exist. If $|P_i| = 2$ at most two clusters exist: one for all genes where the solutions agree and one for where they disagree.

P3 uses crossover to recombine known information in a level of the pyramid with a candidate solution as depicted in Figure 3. For each cluster C_i found by CLUSTER-CREATION, choose a random donor d from the current level such that d's gene values for at least one gene in C_i differs from the candidate solution. Copy all of d's gene values into the solution and evaluate the new solution. If the fitness of the candidate has been reduced, revert the candidate's gene values. It is important to note that candidate changes are kept in the case when the fitness is *unaffected*. This allows crossover to drift, potentially moving through fitness plateaus.

When choosing a donor, P3 ensures there is at least one gene value different between the donor and the candidate solution. This can potentially require up to $O(|C_j||P_i|)$, but empirically appears more efficient than allowing donations that do not change any gene values.

To complete a single crossover event, all of the useful clusters are tested, each requiring at most a single evaluation. Clusters are tested in smallest first order, with clusters of equal size ordered randomly. This ensures smaller building blocks are optimized before large changes are made to the genome, thus helping preserve diversity. This also helps prevent genetic hitchhikers, which may interfere with entropy calculations. As cluster building and application ordering are not dependent on the position of loci in the genome, this operator introduces no positional bias on search.

2.4 All Together

The pieces of P3 were specifically chosen to provide important complimentary capabilities. First, the hill climber provides an efficient tool for discovering pairwise linkage, as search only ends when no single bit change can be made which improves fitness. Therefore crossover, which relies on pairwise linkage, has high quality information to leverage when determining clusters.

Similarly, by starting from a random position for each new solution, P3 progressively reduces the chance of spurious linkage formation due to initialization. Furthermore, these random restarts provide exploration to balance out the hill climber and crossover's extreme exploitation. This also allows the population to grow large enough to contain enough diversity to solve the problem without having to know the optimal size beforehand.

Finally, the use of levels allows P3 to perform solution filtering without throwing away information. Once found, a local optimum is stored indefinitely, preserving both its linkage and genes for future crossovers. By splitting solutions into levels, the linkage information and the donated genes come from similar solution qualities. This prevents noise in gene linkage which only exists in low quality solutions from effecting cluster formation for high quality solutions. It also reduces the chance that optimized solutions will overwrite newer solutions, preserving search diversity.

P3 makes very weak assumptions about the search landscape. In many evolutionary systems the crossover operator assumes linked genes are collocated in the genome. This requires the user to configure genome structure based on gene linkage information they are unlikely to have. All of P3's operations ignore gene ordering, allowing the user to ignore that aspect of configuration.

In general, P3 makes two assumptions about the search landscape: local structure and non uniform gene interdependence. For the hill climber to be effective, there should be at least some correlation between a solution's fitness and the fitness of its single bit neighborhood. Stated another way, this assumes that small changes in genotype result in small changes in phenotype. The non uniform gene interdependence assumption is required by the crossover operator to perform clustering. Without this assumption, the clusters formed by crossover will detect spurious relations. The resulting donation of genes between solutions will be no better than random modification. Both assumptions are likely to hold true for real world problems and are similarly assumed by most evolutionary optimizers.

2.5 Contrasted With The Generational Model

A major advantage of P3's structure is that it avoids the need to set a fixed population size. In a generational model, the diversity contained in the initial population represents a bottleneck for future generations. This is true because in most systems, especially on deceptive problems, mutation rates are too low to create large, multi-gene substructures after initialization. As such the generational model often results in a race between decreasing diversity and increasing fitness.

Premature convergence of a population causes the generational model to have two termination conditions: out of time or out of diversity. In the former, if a smaller population size had been used, less evaluations could have been spent supporting diversity allowing more focus on exploitation. In the latter, if a larger population size had been used, more evaluations could have been performed increasing the likelihood of improving solution quality. In either case, unless the population size is correctly set, the generational model will likely result in waste.

In contrast, P3 has exploited all known information each time the top of the pyramid is reached; no evaluations are wasted on unused diversity. Similarly, because P3 does not throw away previously optimized solutions while adding diversity, it avoids waste caused by premature convergence.

2.6 Order of Complexity Per Evaluation

With the exception of crossover, P3 requires no more than $O(N)$ computations per evaluation, implying it is limited by the time required to evaluate a solution. In order to perform crossover, P3 must spend $O(N^2)$ operations updating pairwise frequencies each time a solution is added to a level of the pyramid. Similarly, it requires $O(N^2)$ time to rebuild the set of crossover clusters once the frequency table has been updated. Naïvely, this leads to the algorithm itself requir-

ing $O(N^2)$ time. However, multiple evaluations can occur for each time a single individual is added to a population.

Each time a random restart occurs, at minimum $N + 1$ evaluations are performed. This means each time a solution is added to P_0, at least $\Omega(N)$ evaluations must be performed. This implies that the $O(N^2)$ cost of the update to P_0 can be amortized over these evaluations, making this update $O(N)$ per evaluation.

To cause an update to P_i, all of the crossovers discovered for P_{i-1} must first be applied, each causing their own evaluation. If P3 were to naïvely use all crossovers and perform an evaluation each time, this would cause $\Omega(N)$ evaluations. Again, this would result in $O(N)$ amortized cost to update P_i. However, Section 2.3 discussed how P3 will ignore some clusters and will also skip an evaluation if no change to the solution was made by a cluster. As such we are currently unable to state conclusively that updating P_i will require $O(N)$ operations per evaluation. That said, empirically it appears to be true. Furthermore, it seems counterintuitive that intentionally wasting evaluations would be considered beneficial to runtime.

There is one more operation in P3 that may not scale linearly with genome size per evaluation: finding a non-identical donor. When performing crossover P3 iteratively searches a population for a random donor with genes that differ from the current solution for the current crossover cluster. In the worst case, this operation could require $|C_j||P_i|$, where $|C_j|$ is the number of genes in the crossover and $|P_i|$ is the number of solutions in the population. However, $|P_i|$ grows very slowly. Also, in empirical evaluation, P3's wall clock time per evaluation appears to scale linearly with problem size.

3. COMPARISON ALGORITHMS

3.1 Random Restart First Improvement Hill Climber

As P3 utilizes a hill climber to perform optimization, it is necessary to show P3's added complexity is able to outperform an optimizer that uses a hill climber alone. Therefore as the first comparison we define a complete optimization method based on the First Improvement Hill Climber from Section 2.1. This definition is extended to match P3's style of restarting. Whenever the hill climber obtains a solution that cannot be improved using a single bit flip, a new solution is randomly generated and optimized.

If P3's quality relies solely on starting at random points and applying FIHC, this comparison algorithm should perform even better than P3. Like P3, the FHIC-alone algorithm is parameter-less, makes no assumptions about gene ordering, and will find the global optimum given a sufficient (potentially exponential) number of evaluations.

3.2 Linkage Tree Genetic Algorithm

The closest relative to P3 in the existing literature is the Linkage Tree Genetic Algorithm (LTGA). There have been a number of different variations proposed to LTGA since its original publication [13], so for comparison we have chosen to use the variant described in the most recent publication [14]. This algorithm is also the current state of the art, outperforming other algorithms in black-box optimization across numerous benchmarks.

LTGA works by iteratively improving a population of solutions in a generational manner. Each generation, half of the solutions (chosen using binary tournament selection) are used to construct a linkage tree just as described in Section 2.3. Each solution in the population is then crossed with the entire population using that linkage tree, with the results put into the next generation. This process continues until the population fails to change between generations, signaling convergence.

Unlike P3, this variant of LTGA does not make use of a hill climber, and there is evidence that doing so results in decreased efficiency [1]. LTGA applies clusters in least linked first order whereas P3 uses smallest first. Also, LTGA does not search for donors until one containing different genes is found. Instead, if the donated genes are identical, it simply skips the evaluation. LTGA requires a population size parameter, maintains a fixed population size across generations, and does not prevent duplicate individuals in a population. LTGA does not add new genetic information after initialization, meaning that if required diversity is not contained in the original population or lost in future generations, LTGA will fail to find the global optimum.

To ensure LTGA is able to find the global optimum quickly, we use the bisection method [4] to minimally set LTGA's population size parameter. To be considered successful, a population size must result in 100 runs without premature convergence. According to the Rule of Three [9], this bounds LTGA's failure rate to less than $\frac{3}{100+1} \approx 3\%$ with 95% confidence. Note that runs used in setting the population are considered testing and were not used during analysis. On randomized problems, training was performed on a disjoint set of problem instances.

Other variants of LTGA have more closely resembled P3 [4], but as those variants do not appear to be in active use, we did not use them for comparison. However, no LTGA variant has removed the population size parameter or fully addressed the problem of premature convergence. There have also been other model based optimization techniques that use few to no parameters. Specifically LO-LIMD [11] uses a population size of one and a parameter free algorithm to perform optimization. It relies on the existence of completely separable subproblems and noise free evaluation in order to determine if two genes are linked through perturbation of their values. While effective on this class of problems, most real world problems do not fall into this category.

3.3 $(1 + (\lambda, \lambda))$

As a final comparison we chose the $(1+(\lambda, \lambda))$ algorithm [2] which is currently the best theory based crossover method. Unlike P3 and LTGA, $(1 + (\lambda, \lambda))$ does not use a population of solutions or any model building to determine problem variable linkage. Instead it uses mutation to produce λ offspring, selecting the best to recombine with the original parent to produce λ more offspring using uniform crossover. In the original work, a method for controlling the value of λ is provided based on offspring success, meaning there are no parameters to be set in this algorithm.

In order to make $(1+(\lambda, \lambda))$ a viable method of optimizing problems with multiple local optima, we modified the original algorithm. Primarily, if during search $\lambda \geq N$, search is restarted from a random individual with $\lambda = 1$. This is done because when $\lambda \geq N$ the mutation rate is greater than or equal to 100%, nullifying search. Furthermore, this point is

only reached when the algorithm has stalled for a significant number of generations.

We also made minor modifications to improve selection and to prevent wasting evaluations, as follows. If the best offspring produced in a generation is a mutant offspring, select that over any of those produced using crossover. If there is a fitness tie between the best offspring in a generation, select the one with the maximum hamming distance to the parent. This encourages drift over plateaus. While the original paper discusses a "mod" version of the λ control strategy for how to handle when offspring of equal fitness are produced, we found this conflicted with our method of restarting. As such we use the original control strategy. Finally, if an offspring produced by crossover has identical genes to either of its parents, it is not evaluated.

4. TEST PROBLEMS

All four of our optimization techniques share some interesting properties for problem equivalence. All are fitness scale invariant, in that changing the fitness values without changing the relation between solution fitnesses will not affect search. None of the methods have a bias toward creating specific gene values, in that XORing all solutions with a fixed string before evaluation will not affect search speed. All of the methods are gene order independent, in that the locations of genes in the genome can be shuffled without effecting search. Combined, these features mean that tests on apparently simple problems can be indicative of search behavior on a large class of actually complex problems.

In choosing test problems, we decided to use only benchmarks with knowable global optimum. This allows us to measure how many evaluations each method requires to converge to the global optimum, as opposed to just the highest fitness reached. This is advantageous as three of the four algorithms have no hard termination condition. The remaining algorithm, LTGA, uses a population parameter which can be increased to slow convergence while simultaneously increasing the fitness reachable before convergence. Therefore it is difficult to say how and when to declare any of the techniques converged unless the global optimum is found.

4.1 Single Instance Problems

Well defined test problem landscapes can help when performing exact analysis of algorithms. These problems can be thought of as testing the edge cases of optimizers, presenting them with unnatural but interesting problem features.

The classic example for this kind of test is the Deceptive Trap problem. For this benchmark the genome is broken up into non-overlapping "traps" of k genes, with the fitness of each trap given by Equation 3.

$$trap(t) = \begin{cases} k-1-t, & t < k \\ k, & t = k \end{cases} \quad (3)$$

In this equation t represents the sum of gene values in the trap. For each trap there is one local optimum (all bits set to 0) and one global optimum (all bits set to 1). We chose to use a trap size of $k = 7$. This problem requires exponential time to solve unless there is some method to effectively move complete traps between solutions, making this an excellent crossover test problem.

However, when using a hill climber and linkage learning, the Deceptive Trap problem may be trivially solved [4]. Thus the more difficult Deceptive Step Trap problem was defined, and is given in Equation 4.

$$step\_trap(t) = \left\lfloor \frac{(k-s) \pmod s + trap(t)}{s} \right\rfloor \quad (4)$$

This problem leverages the original Deceptive Trap problem definition, with the addition of fitness plateaus of configurable size s. This creates an enormous number of local optima and makes detection of linkages much more challenging. We chose to use a trap size of $k = 7$ with plateaus of size $s = 2$.

While both of these problems have difficult to solve subproblems, the non-overlapping nature of the solutions may not capture challenging aspects of real world problem solving. Therefore the Hierarchical If and only If (HIFF) problem is often used to test an algorithm's ability to handle these characteristics [14]. In this problem the genome is broken up hierarchically into a complete binary tree, with each gene representing a leaf, and each internal node representing the union between all genes in that node's subtree. A node in the tree contributes to the fitness only if all of the genes in its subset have the same value (IE are all 0 or are all 1). The amount contributed is equal to the number of bits in the subset. As a result this problem involves creating larger and larger blocks of single value runs which only score if the entire block is optimized.

Our final single instance problem is borrowed from the real valued optimization domain. The Discretized Rastrigin problem uses the standard Rastrigin evaluation function (Equation 5).

$$An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right] \forall x \in [-5.12, 5.12] \quad (5)$$

This function creates a highly multimodal landscape which is completely separable and has a single global optimum. In order to apply our algorithms to this complex problem, we discretized the representation using 10 bit gray coded numbers evenly distributed on the range $[-5.12, 5.12)$.

4.2 Randomly Generated Problem Classes

By enforcing only certain characteristics of a search space while allowing others to be generated randomly, we can perform tests on *classes* of problems instead of specific instances. This can improve the generality of results as experiments then predict the expected quality over the entire class. Unlike single instance problems, the challenge with randomly generated problems is often how to know when the global optimum is reached.

NK landscapes define a class of problems such that the fitness of a single gene's value is dependent on k other genes in the genome. Finding the solution which maximized fitness on an arbitrary NK landscape requires exponential time, and are therefore a good candidate for heuristic techniques. Nearest Neighbor NK landscapes are a subset of NK landscapes such that each gene is dependent on the k genes following it in the genome. The advantage of this subset is that they can be solved in polynomial time [16]. For our experiments we used wrapping neighborhoods, such that genes near the end of the genome depend on genes at the start of the genome. We chose to set $k = 5$.

Ising Spin Glasses are another large class of NP-Hard problems: Given a graph describing the interactions between vertices, set the signs of all vertices to minimize interactions.

Similar to NK landscapes, there is a polynomially solvable subset of Ising problems known as $2D \pm J$ Ising Spin Glasses. This subset restricts graph interactions to a toroidal two dimensional grid with edge weights of $\{-1, +1\}$. This subset can be solved in polynomial time [12], and there are existing tools for solving arbitrary instances.[2]

Our final class is a subset of the Maximum Satisfiability (MAX-SAT) problem. This problem consists of a series of disjunctive clauses, such that each clause contains three problem variables, some of which are negated. While MAX-SAT requires exponential time to solve, we have devised a method of generating only satisfiable instances. First, a random target solution is generated. For each clause, set the signs of each term ensuring the target solution satisfies that clause. To ensure there is no bias to term signs in the generated problem, there is a $\frac{1}{6}$ probability all clause signs match the target, a $\frac{1}{6}$ probability two of the three signs match, and a $\frac{4}{6}$ probability only one sign matches. Note that it's possible for there to be other solutions that also satisfy all clauses. We chose the standard clause to variable ratio of $r = 4.27$.

5. EXPERIMENTAL RESULTS

In total, our experiments performed 100 runs of each optimization algorithm across 7 benchmark problems, with each run limited to 100 million evaluations. In performing data collection, we performed over 150,000 runs, totaling approximately 1.5 trillion evaluations. Of these, approximately 100,000 runs were used to optimally set the population size of LTGA. Note that these runs are considered a training phase, and therefore all results for LTGA use independent runs to create a testing phase. This was the only algorithm that received tuning. As such, LTGA should have a significant advantage over the others in that it has pre-learned the correct population size. All of the code used to perform our experiments as well as individual run results and analysis scripts are available from our website.[3] The results are summarized in Figure 4 and Table 1.

On all 7 problems, the First Improvement Hill Climber and the $(1 + (\lambda, \lambda))$ optimizers appear to have a higher growth complexity than P3. On all of the problems except MAX-SAT they are also worse than LTGA. This makes sense as all of the benchmarks are deceptive in some sense, and neither of these techniques is designed to handle deceptive landscapes. The inability for the iterative hill climber to quickly solve the problems indicates that P3's success is more than just performing local search.

Figure 4 does provide one misleading exception for $(1 + (\lambda, \lambda))$ on the Deceptive Trap problem, where it appears to level out at high genome sizes. This is entirely an artifact of the extremely high variance of this algorithm on this problem. Note that many of the large problem sizes failed to find the optimum for the majority of runs (no data element shown for that size).

The behavior of P3 and LTGA on the four single instance problems appears very similar, with P3 generally being lower by a constant factor. This makes sense as LTGA's tuning allows it to determine a single population size sufficient to solve that instance. However, because it uses a fixed population size, it must set the that size large enough to ensure all random initializations of its population contain enough diversity to reach the global optimum. By comparison, P3 is able to iteratively increase the population size, stopping once enough diversity has been generated. As such we would expect P3 to do better, but not a lot better than tuned LTGA.

	Median	Confidence Interval
Deceptive Trap, N=805, Speedup=2.936		
LTGA	1,807,127	1,805,039 .. 1,808,718
P3	615,503	577,000 .. 653,504
Deceptive Step Trap, N=805, Speedup=1.107		
LTGA	14,106,751	14,102,987 .. 14,109,910
P3	12,742,411	11,951,148 .. 13,777,090
HIFF, N=2048, Speedup=3.1278		
LTGA	1,847,077	1,845,662 .. 1,848,951
P3	590,519	570,739 .. 608,160
Discretized Rastrigin, N=800, Speedup=1.1854		
LTGA	244,086	242,815 .. 245,189
P3	205,908	201,012 .. 210,913
Nearest Neighbor NK, N=600, Speedup=4.408		
LTGA	37,620,027	37,244,617 .. 37,990,772
P3	8,533,322	7,315,275 .. 9,481,271
Ising Spin Glass, N=784, Speedup=20.403		
LTGA	12,677,619	12,601,917 .. 12,761,798
P3	621,347	568,100 .. 667,236
MAX-SAT, N=60, Speedup=942.50		
LTGA	21,130,960	17,576,126 .. 28,314,680
P3	22,420	14,363 .. 29,161

Table 1: Comparison of the median evaluations to success for LTGA and P3. Includes the 95% bootstrapped confidence interval around the median, as well as the speedup factor for P3 over LTGA.

When applied to the randomly generated problem classes, P3 appears to outperform LTGA by an order of complexity. We suspect some of the cause is LTGA's need to fix a single population size. Specifically, during tuning LTGA must set its population size large enough to solve all of the random instances of that class. Therefore all instances which could be solved more efficiently with a smaller population size are penalized. P3 on the other hand grows with problem difficulty. As such its median evaluation time better reflects the median difficulty of a class, whereas LTGA's evaluation time reflects the maximum difficulty of a class.

Of the experiments where LTGA was successful in the median run, it failed to find the optimal solution 109 times, or 0.8%. By comparison, P3 failed to find the optimum 3 times. All three where on Nearest Neighbor NK, two at size 550 and one at size 600. Also, P3 was able to reliably find the optimum on Nearest Neighbor NK $N = 650$, $N = 700$ and MAX-SAT $N = 65$, $N = 70$ while LTGA was not.

Table 1 provides a detailed look at the largest problem sizes in which LTGA was able to find the global optimum reliably in less than 100 million evaluations. For each, we performed a pairwise Mann-Whitney U test, all resulting in p-values less than 10^{-22}, with the exception of Deceptive Step Trap which resulted in 0.00025. As such these results are highly significant. The amount of speedup as compared to optimally tuned LTGA ranged from 1.1 on Deceptive Step

[2] `http://www.informatik.uni-koeln.de/spinglass/`
[3] `https://github.com/brianwgoldman/Parameter-less_Population_Pyramid`

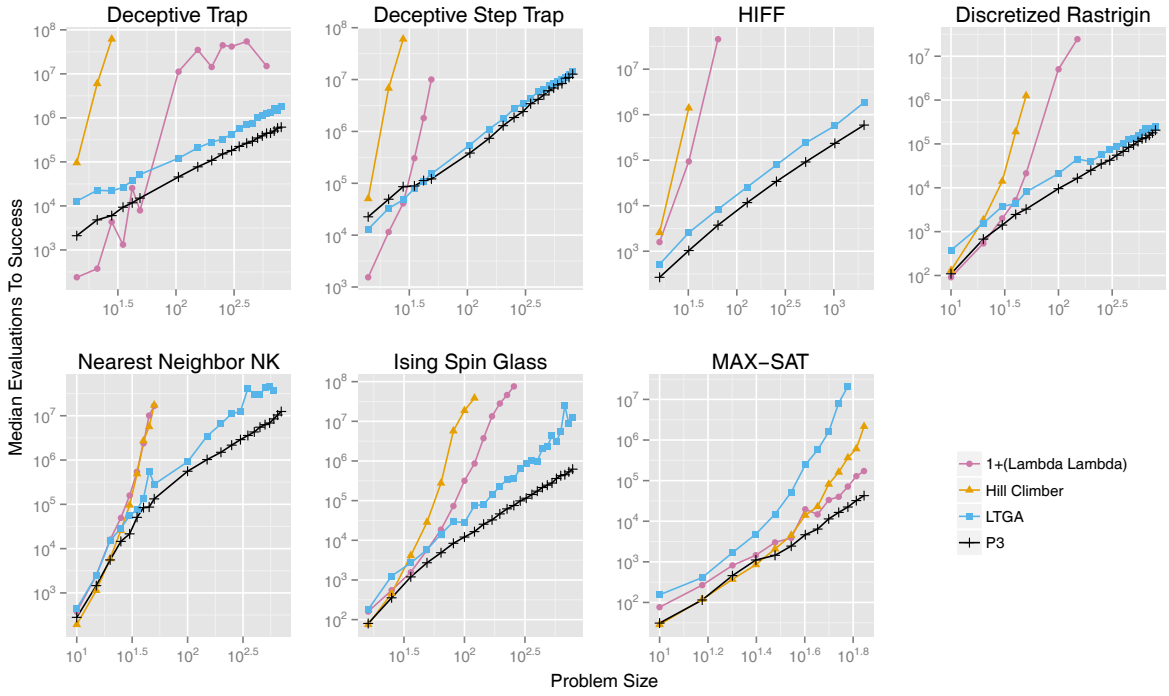

Figure 4: Comparison of the median number of evaluations to reach the global optimum for the four different optimization methods with respect to problem size. If the median run did not reach the global optimum no data element is shown. Results given on a log-log scale.

Trap to 942 on MAX-SAT, but in all cases P3 outperformed LTGA and all other comparison techniques.

As verification that our tuning for LTGA is valid, we refer to [14], which tested the same variant of LTGA that we used. In the section on Child Node filtering, they reported approximately 200,000 mean evaluations to success on the HIFF problem using $N = 512$. Our result of 239,365 is of comparable quality, with P3 solving the same problem using an average of 92,129 evaluations. The discrepancy is likely due to the cited work using their tuning runs in their average.

6. PROMISING THEORY

Due to P3's persistent integration of random solutions, it always has a non zero probability of reaching the global optimum. Because of this (and unlike LTGA), it is possible to derive expected running times for P3 on a number of problems. While we do not present a rigorous solution to any particular problem here, we wish to illuminate potential avenues for future proofs.

The traditional One Max class, as well as all linear functions, will be trivially solved by the hill climber in at most $N+1$ evaluations. These come from the initial evaluation of the random solution, plus the N evaluations required to test each gene for improvement. While more difficult to bound, any unimodal search space without fitness plateaus will be solved by a single application of the hill climber. Notably we expect a bound on Leading Ones of $O(N^2)$ evaluations.

Unlike many theory based algorithms, we believe P3 can achieve bounds on more complex and interesting landscapes. Consider this proof sketch on the Deceptive Trap problem. Each restart will require at most $2N$ evaluations to evaluate the random individual and optimize it to a local optimum using the hill climber. In P_0 each trap in all solutions will either have all 1s or all 0s. As a result $D(C_i, C_j) = 0$ if all C_i and C_j come from the same trap. Therefore all crossover clusters will move only complete traps between solutions, and once there is sufficient diversity in the population there will be a cluster for each individual trap. When performing a crossover of a single trap there can only be a single alternative value, meaning if the candidate solution has all 0s the donor must have all 1s, or vice versa. Therefore if P_i contains the optimum solution for a trap in at least one individual and has a cluster that identifies that trap, all solutions produced by crossing P_i with a candidate solution will have the optimized value of that trap. Extending to all traps we can say that once any P_i has at least one solution with the optimized version of a trap for all traps the very next crossover with P_i will produce the global optimum. The only remaining portion of the proof is to determine how many evaluations are required to reach this point.

Using relaxed requirements, this can likely be extended to any arbitrary, fully decomposable, problem. Assuming that for sufficiently large population sizes the entropy between bits in a single subproblem is lower than the entropy of bits between subproblems, P3 will construct a cluster capable of moving exactly one subproblem's gene values during crossover. As crossover cannot decrease the fitness of a solution each crossover will have a greater than zero probability of improving subproblem solution quality. The probability of low fitness local optima for a subproblem reaching high pyramid levels is therefore low, which increases the proba-

bility of donors only having high quality solutions to subproblems. Combined, this is expected to create a runtime potentially exponential in subproblem size, but polynomial in problem size.

This proof sketch can also be extended to the HIFF problem with minor modifications. Each application of the hill climber ensures all subproblems of size 2 will be solved. Once P_0 has a sufficient number of solutions it will correctly cross all subproblems of size 2 independently, such that all future solutions crossed with P_0 will have all subproblems of size 4 solved. While P_1 can contain solutions that do not correctly solve some 2 bit subproblems (added before P_0 reached critical mass), the probability of these solutions being selected as donors decreases rapidly (solutions added after P_0 reaches critical mass). As such each level effectively serves to optimize the next largest subproblem size, leading to convergence using a logarithmic number of levels.

7. CONCLUSIONS AND FUTURE WORK

While many parameter-less techniques trade the potential quality of optimal tuning for general effectiveness, P3 appears to robustly outperform all of the comparison, state of the art, techniques across a diverse set of benchmark problems. This is because, unlike other parameter-less techniques, P3 does not duplicate effort running multiple populations, nor does it throw away learned information like many evolutionary techniques. By striking a balance between continuously adding diversity and the extreme exploitation provided by the multiple evaluation crossover, P3 is able to solve easy problems quickly and hard problems eventually without any problem specific tuning.

Even when compared against LTGA, the closest relative to P3 in current literature, P3 performs better. This is even after LTGA was extensively tuned to each problem class, an unrealistic proposition on any real world problem.

P3's ability to efficiently optimize not just single problem instance classes, but randomly generated problem classes, without parameter tuning makes it an ideal candidate for real world applications. Its focus on exploiting all known information before spending time increasing diversity makes it a great "stop anytime" algorithm.

While not yet complete, we feel that future work will be able to provide a strong theoretical foundation for the design of P3.

Currently, our results are limited to single objective, noiseless, discrete valued problems. In future work we plan to extend (or show no modification is necessary for) P3 to handle problems from these classes. Doing so will help make P3 more applicable to a wider range of real world problems.

8. REFERENCES

[1] P. A. N. Bosman and D. Thierens. The roles of local search, model building and optimal mixing in evolutionary algorithms from a bbo perspective. In *Optimization by building and using probabilistic models (OBUPM-2011)*, pages 663–670, Dublin, Ireland, 12-16 July 2011. ACM.

[2] B. Doerr, C. Doerr, and F. Ebel. Lessons from the black-box: fast crossover-based genetic algorithms. In *GECCO '13: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 781–788, Amsterdam, The Netherlands, 6-10 July 2013. ACM.

[3] D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise, and the sizing of populations. *COMPLEX SYSTEMS*, 6:333–362, 1991.

[4] B. W. Goldman and D. R. Tauritz. Linkage tree genetic algorithms: variants and analysis. In *GECCO '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 625–632, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM.

[5] J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-16(1):122–128, 1986.

[6] I. Gronau and S. Moran. Optimal implementations of UPGMA and other common clustering algorithms. *Information Processing Letters*, 104(6):205–210, 2007.

[7] G. Harik and F. Lobo. A parameter-less genetic algorithm. Technical Report IlliGAL 99009, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.

[8] G. S. Hornby. ALPS: the age-layered population structure for reducing the problem of premature convergence. In *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 815–822, Seattle, Washington, USA, 8-12 July 2006. ACM Press.

[9] B. D. Jovanovic and P. S. Levy. A look at the rule of three. *The American Statistician*, 51(2):137–139, May 1997.

[10] M. Pelikan and T.-K. Lin. Parameter-less hierarchical BOA. In *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103 of *Lecture Notes in Computer Science*, pages 24–35, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.

[11] P. Pošík and S. Vaníček. Parameter-less local optimizer with linkage identification for deterministic order-k decomposable problems. In *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 577–584, Dublin, Ireland, 12-16 July 2011. ACM.

[12] 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.

[13] D. Thierens. The linkage tree genetic algorithm. In *Parallel Problem Solving from Nature, PPSN XI*, pages 264–273. Springer, 2010.

[14] D. Thierens and P. A. N. Bosman. Hierarchical problem solving with the linkage tree genetic algorithm. In *GECCO '13: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 877–884, Amsterdam, The Netherlands, 6-10 July 2013. ACM.

[15] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[16] 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.