The Multiple Insertion Pyramid: A Fast Parameter-Less Population Scheme

Willem den Besten[1], Dirk Thierens[1(✉)], and Peter A.N. Bosman[2]

[1] Institute of Information and Computing Sciences,
Universiteit Utrecht, Utrecht, The Netherlands
d.thierens@uu.ne
[2] Centre for Mathematics and Computer Science,
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract. The Parameter-less Population Pyramid (P3) uses a novel population scheme, called the population pyramid. This population scheme does not require a fixed population size, instead it keeps adding new solutions to an ever growing set of layered populations. P3 is very efficient in terms of number of fitness function evaluations but its run-time is significantly higher than that of the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) which uses the same method of exploration. This higher run-time is caused by the need to rebuild the linkage tree every time a single new solution is added to the population pyramid. We propose a new population scheme, called the multiple insertion pyramid that results in a faster variant of P3 by inserting multiple solutions at the same time and operating on populations instead of on single solutions.

1 Introduction

The recently introduced Parameter-less Population Pyramid evolutionary algorithm uses a novel population scheme, called the population pyramid [2,3]. A big advantage of the population pyramid is that there is no need for the user to set a fixed population size. Instead, P3 keeps adding new solutions to an ever growing set of layered populations. To explore new solutions, P3 uses the model-building technique and the Gene-pool Optimal Mixing operator from GOMEA [5]. The way it exploits these solutions however, is very different from the classical fixed size population method. P3 is very efficient in terms of number of fitness function evaluations but its run-time is significantly higher than that of GOMEA, this is because P3 rebuilds the linkage tree every time a single new solution is added to the population pyramid, while GOMEA only rebuilds the linkage tree every single generation. A typical GOMEA run only needs about 10 generations, so the difference between the number of linkage trees generated by P3 and GOMEA is huge. We propose a new population scheme, called the multiple insertion pyramid that results in a faster variant of P3 by significantly reducing the number of linkage trees built. Multiple insertion is a population scheme which basically combines P3 with ideas from the Exponential Population Scheme

J. Handl et al. (Eds.): PPSN XIV 2016, LNCS 9921, pp. 48–58, 2016.
DOI: 10.1007/978-3-319-45823-6_5

(EPS) by inserting growing populations instead of single solutions. The multiple insertion pyramid scheme drastically reduces the run-time overhead caused by maintaining a population pyramid. It also requires a lesser number of fitness evaluations than required by EPS [4]. Finally, the multiple insertion pyramid scheme yields a robust GOMEA that performs better in the number of fitness evaluations and on par with the run-time of the original GOMEA on the majority of the test problems. In the next Section, we discuss background information about the Gene-pool Optimal Mixing Evolutionary Algorithm, the Exponential Population Scheme, and the Parameter-less Population Pyramid. Section 3 introduces the Multiple Insertion Pyramid. Section 4 discusses experimental results, followed by our Conclusion.

2 Background

2.1 Gene-Pool Optimal Mixing Evolutionary Algorithm

The Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) is designed for detecting and optimally mixing partial solutions into new solutions [5]. It uses a model, called the family of subsets (FOS), for representing problem variables whose values should be copied together during crossover. Each subset in the FOS is a set of problem variable indices. Each generation GOMEA learns a FOS from the population. Then each solution is subjected to the Gene-pool Optimal Mixing (GOM) crossover operator. First the solution is cloned into an offspring population and for each subset in the FOS, GOM will randomly select a donor from the population and copy the bits indicated by the subset into the current solution. If the donor and the parent solution have the same bits for the problem variables, then the offspring solution is equal to the parent solution and no fitness evaluation is performed. If the donation resulted in an offspring solution with a greater or equal fitness to its parent solution, then the changes are kept, otherwise the changes are undone. All algorithms in this paper use the linkage tree from the Linkage Tree GA as their FOS model [5].

2.2 Exponential Population Scheme

A simple parameter-less population scheme is to keep restarting a GA with an increased population size until the solution quality has become acceptable. The growth function determines the rate at which the population size grows in the number of converged runs. The scheme for doubling the population size is called the Exponential Population Scheme (EPS) [4]. EPS is comparable to the behavior of a user, who will often keep doubling the population size until the results are satisfying. Although there is only a constant overhead, EPS inevitably throws away several populations worth of fitness evaluations.

2.3 Parameter-Less Population Pyramid

The Parameter-less Population Pyramid (P3) uses a novel population scheme, called the population pyramid [3]. The population pyramid is a set of populations with two specific methods for inserting new solutions and performing crossover with the existing solutions. The set of populations expands slowly and forms a pyramid-like structure where the populations are arranged in increasing fitness and decreasing size. Each layer represents a population. Initially there is only one empty population P_0. Every iteration, a new random solution undergoes local search by a first-improvement hill climber and is then added to the bottom layer of the pyramid P_0. Whenever a population has changed due to the insertion of a solution, the linkage tree of that layer is rebuilt. Then the solution undergoes crossover with the layers of the population pyramid, starting with the bottom layer and working its way up. The crossover operator is the GOM operator from GOMEA. If crossover at layer P_i results in a fitness improvement, then the solution is added to the next population P_{i+1} (or a new population is created if there was none existing). The linkage tree at layer P_{i+1} is rebuilt and the solution is GOM crossed over with population P_{i+1}. This process is repeated until the top of the pyramid has been reached. To ensure diversity is preserved duplicate solutions are not added to the pyramid.

P3 selects the donor for the GOM crossover operator differently than GOMEA. GOMEA continues with the next FOS cluster when the solution and donor have equal values for the problem variables. P3 alters this procedure by searching the entire population in a random order until a donor with non-equal values has been found. The upper-bound on the run-time of the crossover operator increases by a factor linear in the population size. Another difference is that P3 sorts the clusters in smallest-first order instead of the randomized order used by GOMEA. However, this has only a minor, problem-dependent, effect on performance.

3 Multiple Insertion Pyramid

The exponential population scheme (EPS) is characterized by its efficiency in model-building, while the parameter-less population pyramid (P3) is characterized by its efficiency in fitness evaluations. From an algorithmic perspective the two algorithms look very different and share only the GOM operator and linkage tree model learning. We now show how to combine the two approaches into a new population scheme. A visual interpretation of the population schemes is given to more easily explain how they are similar and how they can be combined.

3.1 Visualizing Population Schemes

Deriving a generic representation for either population scheme requires a more careful look at how GOMEA works. Each of the schemes relies on the GOM operator for performing crossover between a solution and a population. Two key

properties describe a call to GOM: the set of solutions that are available for donation, and what happens with the solution after crossover.

GOMEA. The grid in Fig. 1a represents GOMEA running for 5 generations (vertical axis) with a population size of 7 (horizontal axis). Each cell of the grid represents the state of solution x at generation y. Each row represents the state of the population at generation y, and each column is the progression of solution x over the generations. The solid arrows show whether the solution changed by GOM was accepted into the next generation. Because GOMEA always accepts the offspring solution into the new population, the progression is a chain of arrows in the visualization. GOMEA traverses the grid in left-to-right first (applying GOM to the population), bottom-to-top second order (initial generation until termination). Lastly, the dashed lines show which solutions from the previous generation have successfully donated bits to a solution.

The probability of a successful donation decreases over time, and this is seen in the uppermost row where the donations are more sparse. If a cell has no dashed line with the previous row, then there was no successful donation during crossover, hence the solution indicated by the cell was not even improved. As seen by the mess of dashed lines in the grid, each solution has the remainder of the population at its disposal for mixing. Let the age of a solution x be the number generations y it has existed, in other words the row index y of column x in the visualization. Two solutions live in the same population if they are improved under the same circumstances, meaning they are improved using the same linkage tree and have access to the same donor solutions. If two solutions live in the same population and are of the same age, then applying GOM crossover to them should be optimal as their fitness should be similar due the competition with each other over the span of multiple generations. This is automatically true for GOMEA, and together with the high availability of donors, the GOM operator is able to optimally mix solutions. GOMEA learns the model, namely the linkage tree, from the entire population (i.e. row) of generation y. Whenever a model is learned before GOM crossover is applied, the corresponding cell is colored gray. For GOMEA this is the start of the generation, i.e. the first column. Clearly GOMEA is minimal in the number of times a model is learned, because otherwise a model would have to be used over multiple generations. No parameter-less population scheme can learn only one model on a row, because then it would have perfectly predicted the population size. Only oracles can tell beforehand what the population size necessary to solve the problem instance is.

The visualization highlights the strengths and weaknesses of a population scheme. Given that the global optimum resides amongst solutions of similar fitness in the final generation, GOMEA has the weakness that it must fully evaluate every generation until the final generation has been reached. However, the sparsity of gray squares shows that relatively few linkage trees are constructed during a single run. The dashed arrows show how optimal mixing allows every solution to donate to every other solution over the entirety of the run.

EPS. Figure 1b shows the visualization of the exponential population scheme (EPS). EPS starts with a run of GOMEA of size one, and continuously restarts

the population on convergence with a doubled population size. Each of the pink squares denotes one run of GOMEA and is traversed exactly as explained in the previous section. The pink squares illustrate how EPS is a rather simplistic wrapper around GOMEA for doubling the population size. An iteration of EPS is the complete evaluation of one run of GOMEA, traversed from left-to-right in the grid. Note that the horizontal axis no longer corresponds to a single population. Learning a linkage tree only uses the solutions of generation y inside the pink square instead of the entire row. The visualization shows the two major weaknesses of the EPS scheme. First, there is no optimal mixing between the runs of GOMEA as indicated by the lack of dashed arrows between the runs. Second, the gray squares are more abundant and show the wasted effort in building linkage trees for the previous generations. But it has already been shown that no parameter-less population scheme can get away with learning only one model per row, hence there being an additional cost of model learning is inevitable.

P3. Figure 1c shows the visualization of the parameter-less population pyramid. With P3 comes a slightly different interpretation which mostly changes how the grid traversed. Green squares indicate that a solution was already present in the population pyramid, hence it was not accepted into the next layer of the pyramid. There is no solid arrow leading into a green square for this very reason. There is a direct correspondence between a population and the layer of a pyramid. Both contain a set of solutions, but the latter cannot contain duplicates by definition. Almost every cell is colored gray to show just how often P3 learns a new model, with the only exception being when a solution is not accepted into the next population. That only occurs when the solution is a duplicate already present in the pyramid, or the solution did not improve in fitness after GOM crossover with the population. The latter happens more often for the higher layers of the pyramid, because improving the fitness of an already optimized solution is more difficult. The linkage tree must be updated every time a new solution is inserted into the corresponding pyramid layer. Each iteration of P3 a solution is drawn from the population generator, and is then improved until GOM crossover has been performed with each layer of the pyramid. A column x is then equivalent to x^{th} iteration of P3. Whereas GOMEA traverses the grid from the left-to-right first, P3 traverses the grid from bottom-to-top first. This behavior is the exact opposite of each other, as is illustrated further in the next subsection. Looking at the dashed lines in the visualization, the solutions receive only donations from the previously evaluated solutions. The mixing that occurs in P3 is unidirectional over the horizontal axis, in contrast to the bi-directional mixing in GOMEA and EPS. Not only that, a solution x at generation y only has access to other solutions that also reached generation y. Hence the number of donors available to a solution (x, y) is lower than the number of donors available to a similarly placed solution (x, y) in GOMEA. Age becomes an even more important factor when GOM is applied to solutions that were improved using different populations. How many the layers the solution has passed (the row) is equivalent to the age of the solution in GOMEA. The linkage trees learned by P3 become more precise as more solutions are added to the pyramid. However, a solution x was improved

with a different linkage tree and population from any other solution. Thus when a solution is donated to, the age of the donor and the receiving solution will be the same, but they were raised in different populations. Hence there is a possible discrepancy in fitness and structure between the solution and the donor, which may lead to lesser performance when performing GOM crossover. The entropy in a population grows slower as more solutions are added, hence the linkage trees will change less over time. As a result, the discrepancy between solutions and donors is reduced over time, but is never truly eliminated. Again, the visualization helps in understanding the strengths and weaknesses of P3. The scheme is over-saturated with model learning at every step of the algorithm, even though the entropy in the populations changes increasingly slower over time. Crossover uses donors of equal age to the receiving solution, but the donors were constructed under different circumstances. This may lead to fitness discrepancies in the early stages of the algorithm, because the model and populations are changing significantly with the addition of a single solution. In exchange for these weaknesses, the population pyramid requires no population size parameter and it gains access to the older generations earlier on than GOMEA.

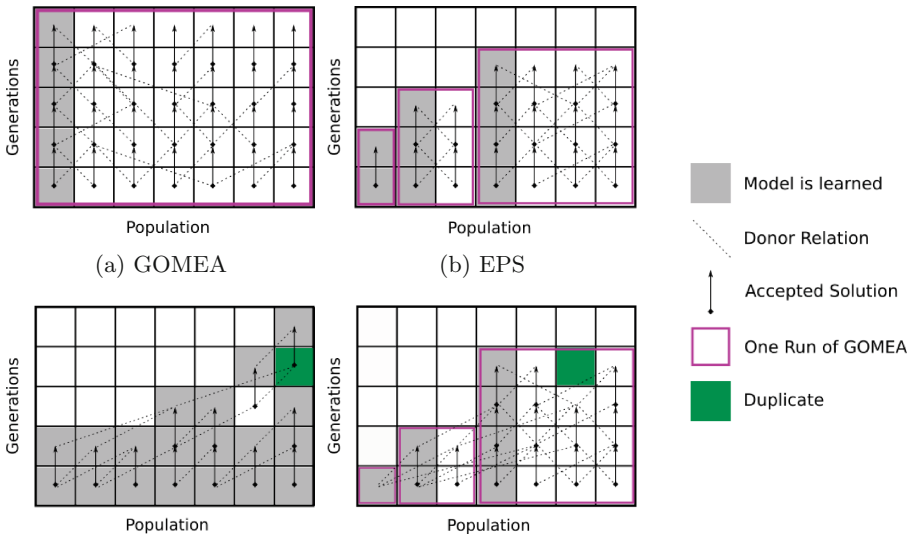

Fig. 1. Visual representation of the four population schemes.

3.2 Multiple Insertion

Multiple Insertion is a new population scheme designed to combine the efficiency in amount of linkage trees built of EPS with the efficiency in fitness evaluations of P3. The goal is to enhance EPS by using the population pyramid as a way to reuse the solutions from the previously converged populations. Similarly, the

goal can be stated as using the strictly growing populations of EPS with P3 instead of one solution per iteration.

As in EPS, Multiple Insertion generates a new, larger population at each iteration. A growth function determines the size of the population generated at the start of iteration, this could be the exponential growth function or the linear growth function. Let P_i be the i^{th} layer (or population) of the pyramid at the start of an iteration. First a population X is randomly generated with the population size determined by the growth function. Optionally the hill climber is invoked on each of the solutions in population X. The entire population X is checked for duplicates already in the pyramid, and the unique solutions are inserted into the bottom layer of the pyramid P_0 and a linkage tree model is learnt. Then GOM crossover is applied to the population X using the pyramid layer P_0. Those solutions whose fitnesses were improved by the GOM crossover are inserted into the set of accepted solutions. Solutions that are already present in the pyramid are filtered from the set of accepted solutions. Finally, the set of accepted solutions is inserted into the next pyramid layer P_{i+1}. This process repeats itself with X and the next pyramid layer P_{i+1} until the top of the pyramid has been reached. Note that the entire population X transitions to the next pyramid layer for GOM crossover, not just the solutions accepted into the next pyramid layer. As with P3, GOM crossover with the top layer may result in a new layer P_t. The new top layer P_t contains every solution from the set of accepted solutions at layer P_{t-1}. Unlike P3, the new layer may contain enough information for GOM crossover to improve the solutions, which would lead to the construction of a new top layer P_{t+1}. This process repeats itself until no fitness improvement is found in the top layer of the pyramid. If none of the solutions were improved by the GOM crossover, then nothing happens to the pyramid.

Visualization. Figure 1d shows the visualization of the multiple insertion population scheme. The density of the gray squares has been significantly reduced in contrast to P3, indicating that there should be a drastic gain in run-time because much less linkage trees are learned. The dashed lines show that solutions can receive donations from either the previous populations or the current population under construction. There is the constraint that a solution is only a donor if it was accepted in the previous generation, because otherwise it would not be in the population. The receiving solution has more available donors during GOM crossover than it would have in either EPS or P3. More importantly, the solutions in the current population are improved using the same model, hence they are of the same age and live in the same population. As mentioned for GOMEA, these conditions allow for optimal mixing between the solutions. When mixing occurs with a donor from a previous population, there may still be a fitness discrepancy as shown for P3. The linkage tree is no longer relearned for every solution inserted into the pyramid, so the population pyramid loses some of its immediate adaptiveness to a change in the pyramid. As more solutions are inserted into the pyramid layer at once, namely a subset of the population currently being improved, the entropy in the pyramid layer changes more significantly than it would have if one solution were inserted. This means that even though the model

is relearned less frequently, the changes to the model are more substantial. The termination criterion for population convergence is a mix between EPS and P3, namely when the population has not been improved during an entire generation and when there are no more layers of the pyramid to perform crossover with.

4 Experimental Results

We have tested five, binary encoded, benchmark problems of increasing length.

1. **Deceptive Trap Function:** we consider the randomly linked DFT which tests the ability to detect and combine good building blocks ($k = 5$).
2. **Hierarchical If and Only If:** we consider the HIFF version where problem variables appear in randomized order.
3. **Ising Spin Glass:** the variables interact with neighboring vertices in a lattice graph. The problem instances are from the public repository hosting the source of the P3 algorithm [2].
4. **Nearest Neighbor NK Landscapes:** each variable depends on the $k = 4$ subsequent variables in the bit-string. The order of the variables in the representation are again assumed to be completely randomized.
5. **MaxCut:** find a maximum cut on a given graph. The instances are taken from [1], and were solved to optimality on the BIQMAC server.

Multiple insertion requires the specification of a growth function to determine how fast the population increases in size per iteration of the algorithm. Let $S(i)$ be the size of the population inserted into the population pyramid at iteration i. Three growth functions are tested: the linear function $S_l(i) = i$, the quadratic function $S_q(i) = i^2$, and the exponential function $S_e(i) = 2^{i-1}$. P3 uses the parameter configuration of the original P3 algorithm. The population pyramid scheme (PT) removes from P3 the hill climber, exhaustive donor searching, and it randomizes the FOS ordering. This way, PT is better comparable with the results from GOMEA. Multiple insertion is added to both population schemes. One can interpret both PT and P3 as a form of multiple insertion with a constant growth function $S(i) = 1$.

Figure 2 shows the relative average number of fitness evaluations for the P3, PT, and its multiple insertion variants. The labels in the legend follow a specific format: {Scheme} – MI – {Growth}, where Scheme is either PT or P3 and Growth is either the constant, linear, quadratic, or exponential growth function. The results are normalized using the results of the original P3 to better highlight the differences in the performance. As seen in Fig. 2 the quadratic and exponential growth functions diverge significantly from PT at several occasions. In contrast, the linear growth function is less prone to this erratic behavior. Especially the exponential growth function tends to have outliers in the number of fitness evaluations, to the point of exceeding EPS. For EPS it makes sense to scale exponentially, as a slower growth leads to more population being converged and rejected, which is terribly expensive. PT with multiple insertion is an extension of EPS that reuses every solution whereas EPS cannot do that. Because PT

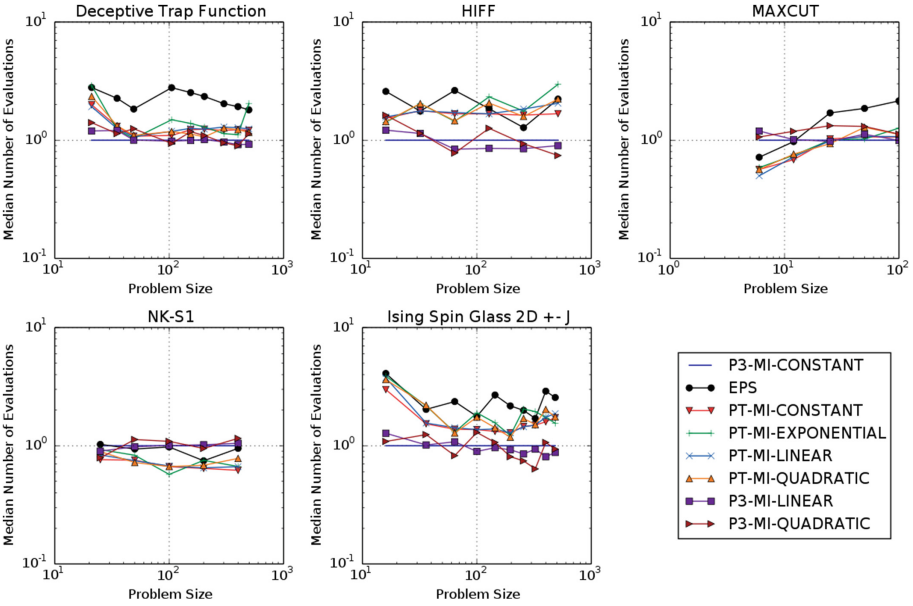

Fig. 2. Relative comparison to P3 of the median number of fitness evaluations for the multiple insertion variants of PT and P3.

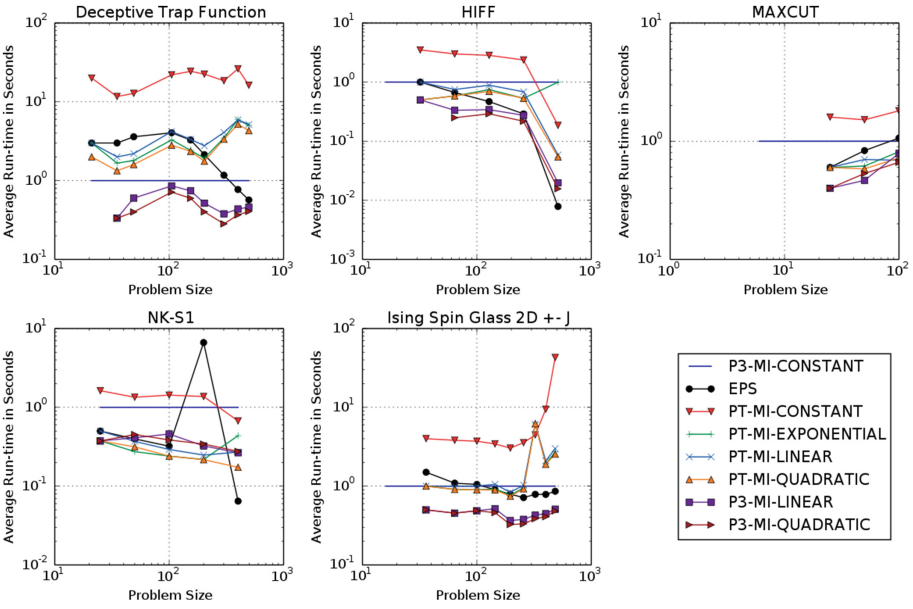

Fig. 3. Relative comparison to P3 of the average running time for the multiple insertion variants of PT and P3.

with multiple insertion is less prone to outliers with a slower growth function, there is little reason to use an exponential growth function.

Figure 3 shows the relative average run-time in milliseconds for P3, PT, and the multiple insertion variants. Using multiple insertion with PT reduces the run-time to the level of the original P3 which uses the hill climber. However, increasing the growth function beyond linear has no significant effect on the run-time, as seen by the tightly grouped lines per population scheme. Applying multiple insertion to the original P3 algorithm greatly speeds up the run-time. On any of the test problems the run-time of P3 matches or outperforms EPS, which is something it rarely achieved without multiple insertion. The exception to the rule are the NK-S1 problem instances, but that is due to the poor performance caused by the hill climber. The effectiveness of multiple insertion shows that P3 wastes a liberal amount of processing time on rebuilding linkage trees to little effect. The results also indicate that the performance of the linear growth function is on par with the quadratic function and more reliable than the exponential function, hence we recommend the use of the linear growth function.

5 Conclusion

We have proposed a novel parameter-less population scheme, called the Multiple Insertion Pyramid. We have discussed two populations schemes that do not require the user to set a fixed population size: the Exponential Population Scheme (EPS) and the Parameter-less Population Pyramid (P3). P3 is a model-based evolutionary algorithm that applies the linkage tree model building and the Gene-pool Optimal Mixing crossover operator from the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA), but uses a novel population scheme as exploitation mechanism. P3's population pyramid scheme outperforms the linkage tree based GOMEA in terms of number of fitness evaluations at the cost of a significantly increased run-time. It is noted that P3's method to rebuilt the linkage tree whenever a single new solution is added to the population pyramid is too costly. By changing the population scheme to the Multiple Insertion Pyramid scheme the run-time is significantly reduced. The Multiple Insertion Pyramid combines EPS with the population pyramid, which allows solutions from previously converged populations to contribute to the current population. Because previous populations are retained for donation, the growth rate is reduced from exponential to a linear function. Multiple insertion can be applied to P3 to greatly reduce the time spent on relearning linkage trees.

References

1. Bosman, P.A., Thierens, D.: More concise and robust linkage learning by filtering and combining linkage hierarchies. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 359–366. ACM (2013)
2. Goldman, B.W., Punch, W.F.: Parameter-less population pyramid. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 785–792. ACM (2014)

3. Goldman, B.W., Punch, W.F.: Fast and efficient black box optimization using the parameter-less population pyramid. Evol. Comput. **23**(3), 451–479 (2015)
4. Harik, G.R., Lobo, F.G.: A parameter-less genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 258–267 (1999)
5. Thierens, D., Bosman, P.A.: Optimal mixing evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 617–624 (2011)