Logarithmic-Time Updates in SMS-EMOA and Hypervolume-Based Archiving

Iris Hupkens and Michael Emmerich

Leiden Institute of Advanced Computer Science, Faculty of Science, Leiden University, 2333-CA Leiden, The Netherlands
{ihupkens,emmerich}@liacs.nl

Abstract. The hypervolume indicator is frequently used in selection procedures of evolutionary multi-criterion optimization algorithms (EMOA) and in bounded size archivers for Pareto non-dominated points. We propose and study an algorithm that updates all hypervolume contributions and identifies a minimal hypervolume contributor after the removal or insertion of a single point in $\mathbb{R}^2$ in amortized time complexity $O(\log n)$. This algorithm will be tested for the efficient update of bounded-size archives and for a fast implementation of the steady state selection in the bi-criterion SMS-EMOA. To achieve an amortized time complexity of $O(\log n)$ for SMS-EMOA iterations a constant-time update method for establishing a ranking among dominated solutions is suggested as an alternative to non-dominated sorting. Besides the asymptotical analysis, we discuss empirical results on several test problems and discuss the impact of the overhead caused by maintaining additional AVL tree data structures, including scalability studies with very large population size that will yield high resolution approximations.

1 Introduction

In evolutionary multiobjective optimization algorithms *(EMOA)* [6] and algorithms that keep bounded-size archives for Pareto optimal points [14] (bounded-size archiving), fast ranking schemes that take into account diversity and proximity to Pareto fronts are required. The hypervolume indicator (also: hypervolume, S-Metric, Lebesgue measure) is a commonly applied measure for the quality of a Pareto front approximation. As it can be computed without knowledge of the true Pareto front, it can be used not only in performance assessment, but also in selection schemes of EMOA and in bounded-size archiving. The maximization of the hypervolume indicator over bounded-size approximation sets, yields approximation sets that are distributed across the Pareto front and cover a diverse set of attainable objective function vectors (see Fonseca and Fonseca [5])[1].

The focus of this work will be on EMOA and bounded-size archivers that use single point replacement schemes to keep the size of a population bounded. After

[1] An objective function vector is attainable, if it is either possible to find a preimage of this vector in the search space or a preimage of a vector that dominates it.

M. Emmerich et al. (eds.), *EVOLVE - A Bridge between Probability, Set Oriented Numerics and Evolutionary Computation IV*, Advances in Intelligent Systems and Computing 227,
DOI: 10.1007/978-3-319-01128-8_11,

a point is added to a population or archive of size n, the size of this population is kept constant by selecting a subset from the now $n+1$ points that maximizes the hypervolume indicator. This is equivalent to removing a point with minimal exclusive contribution to the hypervolume indicator (hypervolume contribution). Single point replacement is used in bounded size archivers [14] and also in the SMS-EMOA [8] which is one of the first and most commonly applied EMOA using hypervolume-based selection.

A disadvantage of using hypervolume indicators in selection schemes has so far been, that as compared to simpler selection schemes the required computational overhead caused by repeated computations of hypervolume indicators is high. This is in particular the case for many-objective optimization problems, but also in the very common case of bicriteria optimization hypervolume indicators where hypervolume contributions can be computed with time complexity in $O(n \log n)$. In particular when working with fast-computable objective functions and large population sizes and number of iterations, hypervolume computations can dominate the computational effort and strongly limit the algorithms' performance.

The goal of this paper is to provide provably fast incremental algorithms for implementing the hypervolume-based selection in SMS-EMOA and online updates of hypervolume-based bounded archiving. The focus is on bi-criterion problems, and we will study both asymptotical time complexity and empirical performance.

In Section 2 some of the concepts used throughout this paper are explained. Section 3 describes SMS-EMOA, and some other work related to this paper. In Section 4 we prove that updating hypervolume contributions is possible with amortized time complexity of $O(\log n)$ by presenting an algorithm that proves that theorem. In Section 5 we discuss how this theorem can be used to improve the time complexity of SMS-EMOA. In Section 6 the performance of this proposed method is investigated by measuring the time needed to update hypervolume contributions in an implementation of a 2-D archiver which uses the proposed update scheme, and the empirical performance of a fast SMS-EMOA implementation is tested. Finally, Section 7 contains some concluding remarks.

2 Technical Preliminaries

Here we consider minimization of k objective functions $f_i : X \to \mathbb{R}$, $i = 1, \ldots, k$ over some decision space X. In particular we will focus on the case $k = 2$ (bi-criterion optimization), but some techniques can be generalized to more dimensions. Most of our discussion will be on finite point sets in the objective space $\mathbb{R}^k$ that contains the image vectors (or: objective vectors) of the mapping provided by $(f_1, ..., f_k)^T$. As selection and archiving procedures operate on sets of objective vectors, it is of no concern for the discussion of these algorithms where the preimages of these vectors are located. We will use the notation $p.x$ to denote the f_1 coordinate of an objective function vector p and $p.y$ to denote the f_2 coordinate of an objective vector p. In this case Pareto dominance is defined

as follows: An objective vector p dominates an objective vector p', if and only if $p.x \leq p'.x$ and $p.y \leq p'.y$ and $p \neq p'$.

Each objective vector in a bi-criterion optimization problem can be visualized as a 2-dimensional point in the space spanned by the values of f_1 (horizontal axis) and the values of f_2 (vertical axis).

See Figure 1. Green points (stars) depict non dominated objective vectors, blue points denote dominated solutions, and the red curve separates the attained subspace from the non-attained subspace.

In Pareto optimization with bounded-size archives/populations we are interested in finding a set of n non-dominated solutions that approximate a Pareto front in the objective space. A set of n objective vectors will be termed an approximation set of size n. Here the Pareto front is the set of all non-dominated objective vectors that can be obtained from solutions in X.

It is usually infeasible to calculate the entire Pareto front. The amount of solutions in it might be infinitely large, or simply too large to be practical. Therefore it becomes important to find an approximation set without knowing the true Pareto front. In this case a method for measuring the quality of a finite approximation set is needed. The hypervolume indicator is one such measure.

2.1 Hypervolume and Hypervolume Contributions

The dominated hypervolume of a set of points representing solutions is the area (or in arbitary dimensions the Lebesgue measure) of the set of all points in the objective space that is dominated by that set and bounded from above by a reference point. In Figure 1, the area above the attainment curve is the dominated hypervolume of the Pareto optimal set. The higher the dominated hypervolume of a set of points, the better an approximation it is considered to be. The dominated hypervolume contribution of a point in this set represents the volume of the search space that is dominated by that point, and not by any other points in the set. If a point needs to be removed from the set to make it smaller, the point with the lowest hypervolume contribution is a good candidate, as this removal is equivalent to maintaining the subset of size $n - 1$ which dominates the largest hypervolume.

2.2 AVL Trees

An AVL tree [11] is a binary search tree that keeps an internal balance factor for every node, representing the difference in height between its left and right subtrees. The AVL tree is considered balanced if this difference is not larger than 1 for any node in the tree, in which case its height is bounded by approximately $1.44 * \log n$. Nodes on the path from a newly inserted/deleted node to the root are rotated if the imbalance exceeds 1. Because the height of the AVL tree is bounded, and because all overhead required to keep it that way has a complexity of $O(\log n)$, the lookup, insertion and deletion operations on an AVL tree all have a worst-case complexity of $O(\log n)$. Red-black trees have been suggested

as an alternative to AVL trees, but essentially offer the same complexity for the operations considered in this work.

3 SMS-EMOA and Related Work

SMS-EMOA [8] is an evolutionary multiobjective algorithm that uses the hypervolume measure as selection criterion. In each iteration of the algorithm one new individual is created by using random variation operations. After adding the new individual, an individual with minimal contribution to the hypervolume indicator is removed. All dominated solutions have a contribution of zero. Therefore, in order to decide among multiple dominated solutions which one to delete, SMS-EMOA partitions the approximation set into layers of mutually non-dominated points (fronts) of decreasing rank and deletes an individual on the worst ranked layer. Calculating the differently ranked fronts is done using the fast-nondominated-sort algorithm from NSGA-II [7], and has a worst-case complexity of $O(mn^2)$ with m being the number of dimensions. When there is only a single front and the problem is two-dimensional, calculating the hypervolume contributions of the points in that front is dominated by the sorting algorithm and has a complexity of $\Theta(n \log n)$. Calculating the hypervolume contribution of any one point can be done in constant time plus the time it takes to find the points that come right before and right after it in the sorting order, and therefore takes only $O(n)$ time if the points are already sorted.

Bringmann and Friedrichs [3] show that not just calculating, but also approximating the least hypervolume contributor is #P-Hard in the number of dimensions d. If d is constant, the problem has polynomial complexity. They have also devised an algorithm for calculating the hypervolume contributions of sets in higher dimensions[4] which runs in $O(n^{d/2} \log n)$. For two and three dimensions calculating the minimal hypervolume contributor and all contributions has time complexity $\Theta(n \log n)$ [9].

Other work that has been done in the area of updating hypervolume contributions quickly includes [2], in which a method is given for not having to fully recalculate the hypervolume contributions of points in three dimensions by keeping them the same if they can be shown to not have changed, and by recalculating hypervolume contributions after the removal of a point based on the contributions of that point (which is usually smaller and simpler to calculate since it is the worst point). No complexity analysis is given for this method, though a large empirical speedup is shown during experiments.

4 Logarithmic Time Updates

In this section, we will prove the following theorem by providing a complexity analysis of an algorithm that implements it:

Theorem 1. *Let P denote a set of n mutually non-dominated points on a 2-D plane. When a new point is inserted into P, all points dominated by that point*

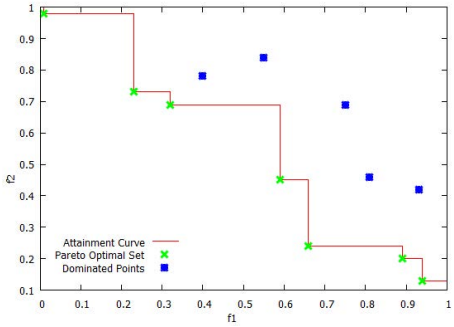

Fig. 1. A visual representation of a set of objective vectors. Blue squares: dominated points. Green stars: Non-dominated points.

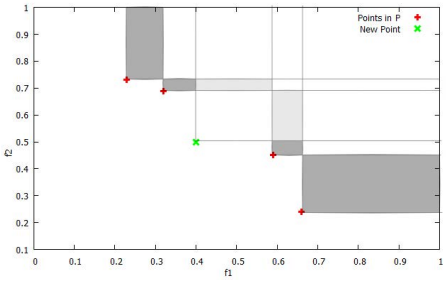

Fig. 2. A visualization of the hypervolume contributions of a set of mutually non-dominated points, and how the insertion of a new point affects them

are removed. Updating P and the hypervolume contributions of the points in P after adding a single point can be done with a time complexity of amortized $O(\log n)$.

Proof. Points in P are kept in an AVL tree, sorted in order of ascending x value. Because the points in P are mutually non-dominated, each point that comes after a point in the sorting order will need to have a smaller y value than that point to not be dominated by it. Therefore, the set of points is implicitly also sorted in order of descending y value.

Since the points are on a 2-D plane, the hypervolume contribution of a point p in P is equal to the rectangle which has p at its lower left corner, with the height being equal to the difference in y value between p and the point in P with the next lowest y value and the width being equal to the difference in x value between p and the point in P with the next lowest x value. See Figure 2 for a visualization. The size of the hypervolume is also bounded by a reference point representing the 'worst possible' point in the search space. This only affects the hypervolume contributions of the two outermost points, which lack a neighboring point with a lower x or y value that would otherwise bound their hypervolume contribution. In SMS-EMOA we always want to keep those points, and therefore we can sidestep the problem of choosing a reference point by setting the hypervolume contribution of these points to be infinitely large. Whenever we have a point p which is not part of P, but which we are considering adding to it, we have to perform the following operations:

1. *Check if p is non-dominated by the points already in P by finding a neighbor in the x value*: If it is dominated, then we do not want to include it, so we can skip the following steps. Only the point q with equal or next lower x value to $p.x$ needs to be considered, because points s with $s.x > p.x$ can never dominate p, and if q does not dominate p, no other point can dominate it

since all the points s with $s.y < q.y$ have $s.x > p.x$. In an AVL tree, finding q can be done in $O(\log n)$ in the worst case. Once the point has been found, checking domination can be done in constant time by comparing y values. If $q.y \leq p.y$, the candidate point is dominated and can be discarded. There is also the possibility of both x and y being the same: in that case, p and q are the same point, and all of the following steps can also be skipped.

2. *Insert the point into P.* Inserting a point into an AVL tree can be done in $O(\log n)$. If $p.x = q.x$, for q found in the first step, p should be inserted in the position preceding q in the sorting order. While this temporarily violates the implicit sorting in the order of descending y value, it causes q to be properly deleted in the next step.
3. *Delete points that are dominated by p.* Points with a lower x value will never become dominated, so only the points that occur later in the sorting order have to be considered, and only their y value has to be compared to $p.y$. Since the points are sorted in descending order of y value, we can start at the point which occurs after p in the sorting order, process points in descending order until we have either reached the last point in P or found a point that is not dominated by p, and delete the interval of points in between p and that point.

   Deleting a single point can be done in $O(\log n)$, because that is the worst-case complexity of both deletion and lookup in an AVL tree. In the worst case, up to n points will be deleted, so there is an upper bound on the worst case complexity for this step of the algorithm of $O(n \log n)$.
4. *Update the hypervolume contributions of points in P.* When a point is added to or removed from P, up to three points need to have their hypervolume contribution changed: the point that was inserted, and the points that come right before and after it in the sorting order. Note that it is not necessary to update the hypervolume contribution after deleting a point that is dominated by p, as done in step 3, because, since the points that are deleted are always in a row, the neighbors of a point that was just deleted are either the same neighbors that will need to be updated due to the insertion of p, or points that are themselves dominated by p.

   The calculation of the new hypervolume contribution of a point can be done in constant time, but in the worst case $O(\log n)$ time can be required to find a neighbor point in the AVL tree. Therefore the worst-case complexity of this step is also $O(\log n)$.

The worst-case complexity of adding a point to P stems from the complexity of the second operation described above, in which points are deleted from P. It is the only step which has a complexity greater than $O(\log n)$. However, each point can only be deleted from P once, and the number of points in P can only increase (by one) if a point is added without deleting any points. If this update scheme is used to add every point to the set, then for every k points that are deleted there have to have previously been at least k updates of the set which did not require the deletion of any points. Therefore, the amortized complexity of this update scheme is on average $O(\log n)$ per update after any n steps of the algorithm.

A few shortcuts naturally present themselves when implementing the proposed update scheme. It is usually not necessary to find the neighbors that are used in step 4, because they are already found in previous steps: if the right neighbor exists, it is always the point that was found in step 3 as the first point that is not dominated by p. The upper neighbor is point q as found in step 1, unless the x value of q and p was equal. In that specific case, the true upper neighbor does need to be found in the AVL tree before being able to update the hypervolume contribution of p, but its own hypervolume contribution will not need to be updated since the only part of the right neighbor that is used to calculate the hypervolume is its x value, which stayed the same.

The update scheme described can be extended to allow for removing points with the lowest hypervolume contribution from P:

Corollary 1. *A replacement is a transaction consisting of an insertion of a new point to a set P in $\mathbb{R}^2$ and subsequent removal of the point in P with lowest hypervolume contribution. The amortized time complexity for n replacements is $O(\log n)$ per replacement after any n replacements.*

Proof. A second AVL tree can be maintained that is sorted by hypervolume contributions. This tree will be called the *HC-tree*, while the tree that is sorted by x values will be called the *X-tree*. The HC-tree needs to be updated (with complexity $O(\log n)$) whenever a hypervolume contribution is changed, a point is deleted, or a point is inserted, each of which already requires an operation on an AVL tree with a complexity of $O(\log n)$. When this second AVL tree is maintained, finding the point with the worst hypervolume contribution can be done with a single lookup of complexity $O(\log n)$. Deleting this point can then be done in $O(\log n)$, including the necessary updates of the hypervolume contribution of the point that comes right before it in the sorting order and the point that comes right after it.

Remark 1. By extending the update scheme in this way, it is possible to implement a *bounded 2-D archiver* which works with amortized time complexity $O(\log n)$, where n is the maximum size of the archive.

It is often desirable to maintain the value of the hypervolume indicator for an archive, in order to measure progress:

Corollary 2. *Starting from the empty set and a stream of points to be inserted one by one to the archive, online updates of the hypervolume indicator can be computed in amortized $O(\log n)$ time per point.*

Proof. One possibility to achieve updates of the hypervolume indicator in 2-D in logarithmic time is to keep track of the cumulated hypervolume contributions in the above algorithm. Another possibility, is to use a modified version of the dimension sweep algorithm of Fonseca et al. [1,10] for computing the hypervolume indicator in 3-D. This proceeds in increasing order of the third (or z) value of the objective function vector and successively updates the area dominated by a set of points in their projection to the xy-plane. For any n such updates the

algorithm requires time in $O(n \log n)$. The modification is as follows: One can simply use the update scheme of the hypervolume indicator in the projection to the xy-plane. (This way, the z value can be interpreted as the time when a point enters the archive to be minimized.)

Remark 2. Using the second idea in the proof of corollary 2, the dimension sweep algorithm for computing the hypervolume indicator in 4-D that was proposed by Guerreiro et al. [12] can be used for an amortized $O(n)$-time incremental update scheme of the hypervolume indicator for tri-criterion optimization. In this case points are inserted in the order of the fourth dimension into a 3-D hypervolume-based archive of point projections to the xyz-hyperplane. After n updates the amortized time complexity is $O(n^2)$. A more detailed analysis of the higher dimensional cases and unbounded archiving is beyond the scope of this paper and left to future work.

5 Improving SMS-EMOA

SMS-EMOA keeps a population which includes not just mutually non-dominated points, but dominated points as well. While storing all dominated points in an additional data structure after they are removed from P is a trivial addition, SMS-EMOA requires the hypervolume of the worst-ranked front to be calculated rather than the best-ranked front. It is possible to use the update scheme described above to update all the fronts, however this comes with an increase in complexity. The situation shown in Figure 3 illustrates that there can be a large number of changing ranks as a consequence of the insertion of a single point. This can reoccur an arbitrary number of times through the runtime of the algorithm.

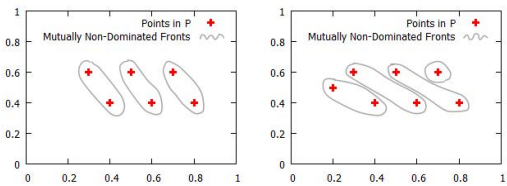

Fig. 3. If dominated fronts are kept in sets of mutually non-dominated points, then inserting a single point can necessitate recalculating all sets - and all hypervolume contributions

As a fast alternative method it is proposed to rank the dominated points based on how long ago they were removed from the set of non-dominated points. If only small numbers of points are dominated, then the difference in ranking caused by not calculating hypervolume contributions will be marginal. In situations where many points are dominated, though, the modification of the scheme might have a bigger impact. It will thus be tested in the next section.

6 Empirical Performance

All empirical tests were performed on the same computer and using the same compiler: A GNU C++ compiler, O3 optimized, using MINGW/Windows 7 on an Intel i7 quadcore CPU with 2.1 GHz clockspeed and 8Gb RAM. For time measurements a stopwatch method based on the processor's high resolution performance counter was used. Source codes are available on request to the authors.

In order to test the performance of the proposed update scheme, a bounded-size 2-D archiver was implemented. This archiver only calculates the hypervolume contributions of points that are not dominated by any other points in P. Dominated points are stored in a double-ended queue, from which the oldest point is removed whenever room needs to be made for a new point.

6.1 Performance of Incremental versus Non-incremental Hypervolume Updates

The performance of the incremental and non-incremental update schemes using AVL trees was tested using the following method: Pseudorandom numbers used in the tests below were generated using a 64-bit linear congruential generator of which only the 32 highest-order bits were used. In a first test, each n, a set P of n different points was prepared for which the x value was an even pseudorandom number between 0 and 2^{31}, and the y value was equal to $2^{31} - x$. This choice of values causes all possible points to be mutually non-dominated or the same point. Hypervolume values were computed in 64-bit. Then, 100000 insertions were performed by repeatedly generating a point p for which the x value was an odd pseudorandom number between 0 and 2^{31} and the y value was again equal to $2^{31} - x$, then inserting p into P and removing the point which had the worst hypervolume contribution afterwards. The insertion and the removal that resulted from it were included in the time measurements, the preparation of P and the generation of the random numbers were not.

Figure 4 shows the results averaged over 20 runs as compared to a non-incremental update scheme. The non-incremental scheme recomputes all hypervolume contributions in each iteration, but already keeps the population sorted using an AVL tree.

The linear time complexity of the non-incremental hypervolume update scheme is especially noticeable, as it almost immediately dominates the logarithmic time complexity of the AVL tree operations necessary to insert and remove points from the set. For the implementation used in this test, an incremental update scheme starts outperforming a non-incremental update scheme when P is bounded to values higher than about 70.

To better visualize the time complexity of the incremental hypervolume update scheme, the test was also performed for larger values of n, where n started at 4 and was repeatedly multiplied by 1.5. Figure 5 shows the results of this test averaged over 20 runs. With logarithmic time complexity, the expected result was a diagonal line when plotted logarithmically, however this was not the case.

The time taken increases slowly for a while, then increases faster, then starts to increase more slowly again.

Examining the operations performed during the tests shows the likely cause. Figure 6 shows the number of rotations performed in each of the AVL trees during the test; again, averaged over 20 runs, and there was little variance. For every point that is removed from the X-tree, up to three points need to be removed from the HC-tree and up to two points need to be inserted, and for every point that is inserted into the X-tree, up to two points need to be removed from the HC-tree and up to three points need to be inserted. Therefore, the number of rotations can be five times as large.

When n is very small, the limited amount of possible structures that an AVL tree with n points can have causes the values to jump around a bit at first before settling into a pattern. The number of rotations stays much the same both when inserting and deleting points in the X-tree. The HC-tree behaves differently. The rotations required after deleting a point in the tree shrinks slowly for the first part, then starts rising and reaches its peak after n has reached six digits. In the timing graph, the slope of the HC-tree deletion rotations is noticeable in the slope of the timing of the test as a whole. It is flatter in the part where the HC-tree deletion rotations are slowly decreasing, then becomes steeper when the HC-tree deletion rotations start to increase.

Once n is above ca. 10^6, the effect of the HC-tree deletion rotations is replaced by that of the HC-tree insertion rotations. These initially stay pretty much constant, just like the X-tree rotations, but once the HC-tree deletion rotations become more common, this type of rotation starts to become more numerous as well. The amount of HC-tree insertion rotations increases very quickly for a while after the HC-tree deletion rotations start going down again. Due to memory limitations, it was not feasible to test whether the HC-tree insertion rotations stay at this new level or if they will gradually decrease in number after this point.

The behavior of the HC-tree might be a consequence of how it is used. Nodes are not equally likely to be inserted or removed. The leftmost node is removed every time, as it represents the hypervolume contribution of the node that is removed to keep the size of P constant. The other nodes that are removed from the HC-tree represent neighbors of a point that was inserted or removed, meaning they will be replaced by nodes that are smaller or larger respectively, and thus more likely to end up in the sides of the tree than in the middle. Either way, the value of n for which the amount of rotations starts increasing is sufficiently large for this to be of no concern for practical purposes, and even after it increases the algorithm is still fast.

Next, initialization time or time for incrementally updating an unbounded archive was empirically evaluated. See Figure 7 for a plot of the time taken to either fill an AVL tree with n points (for the non-incremental update scheme), or fill an AVL tree with n points and incrementally update the hypervolume contributions of each of these n points after each insertion of a point (for the incremental update scheme). As can be seen, the incremental update scheme

has an almost linear increase in a linearly scaled diagram. This shows that the logarithmic factor in the $O(n \log n)$ time complexity for n insertions is almost negligible. Used in the initialization phase of an unbounded archiver, the incremental scheme would take several times longer than a scheme using only a X-tree. For both methods, the cost of initializing, even for large archives is still only in the range of milliseconds. Note, that the cost of not using any tree would be prohibitively high for the population sizes reported in the diagram.

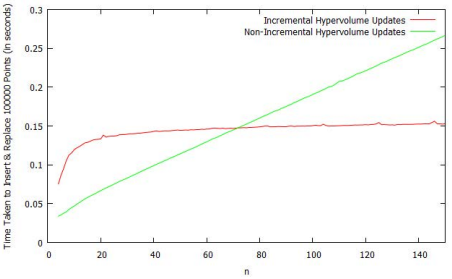

Fig. 4. Performance of the update scheme described above, versus an non-incremental hypervolume update scheme

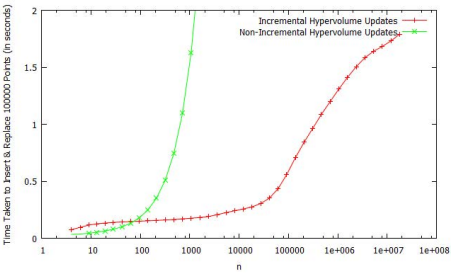

Fig. 5. Logarithmic plot of the performance of the incremental and non-incremental hypervolume update schemes

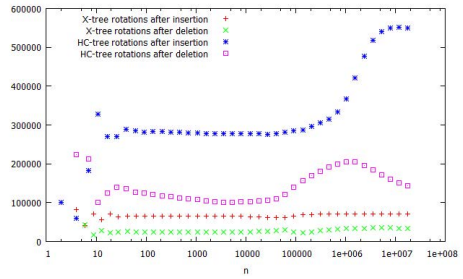

Fig. 6. Average number of rotations required for AVL tree updates during each test

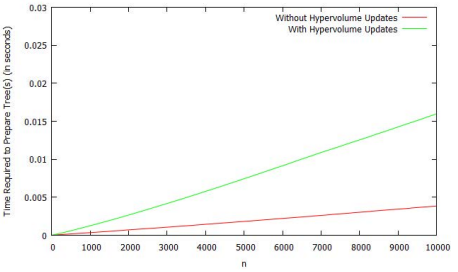

Fig. 7. A plot of the time taken to prepare an initial population of n mutually non-dominated points (averaged over 20 runs)

6.2 AVL Trees versus Non-self-balancing Binary Search Trees

Although the complexity of a regular binary search tree is $O(n)$ in the worst case, this cannot be expected to occur in the average case, and there is some overhead associated with the self-balancing of the AVL tree. To measure whether this overhead is significant, the first test described in Section 6.1 was repeated, except the incremental update scheme was compared to a version of itself without the self-balancing parts of the AVL tree code (turning the trees into regular binary search trees). As seen in Figure 8, the average performance of the AVL tree is

very similar to that of the regular binary search tree. On average, it appears to get slightly slower with increasing n. Not pictured: the non-self-balancing binary search tree starts rising much more sharply after n becomes larger than about a million. That result is consistent with the results found in Section 6.1, where the number of rotations required to keep the HC-tree balanced after insertions increases after that point. However, it needs to be investigated still whether this comparison provides the same results for less symmetrical test problems.

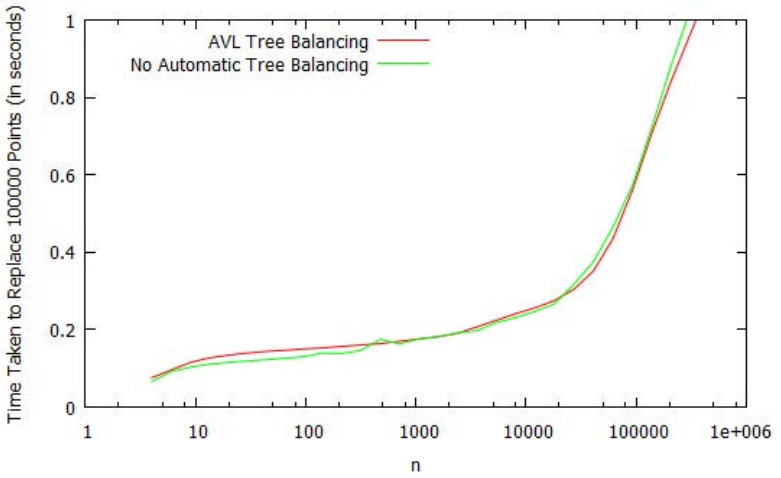

Fig. 8. A plot of the average time taken to insert 100000 points into a set of mutually non-dominated points, with or without a self-balancing AVL tree (averaged over 20 runs)

6.3 Fast SMS-EMOA

Finally it was tested whether the update scheme can be applied in order to speed-up SMS-EMOA. For the classical 2-D test problems ZDT1, ZDT2, ZDT3 and ZDT6, a number of 10 runs per problem were performed for (1) the canonical SMS-EMOA described in Emmerich et al. [8] in its original implementation and (2) for an SMS-EMOA using fast incremental updates and a queue to maintain dominated solutions. Settings were as in [8], e.g. population size was 30 and 20000 iterations were performed. The reference point was $\{10000000\}^2$. The resulting histories of the hypervolume indicator are depicted in Figure 10 to Figure 17. As can be seen from the graphs, the changed selection scheme does not lead to worse results. Also by visual inspection of a final result, it was observed that both methods achieve very accurate results (cf. Figure 9) However, instead of taking a total time of 1.585 seconds for all selection steps, a single run with fast SMS-EMOA now takes only 0.02936 seconds for all selection steps. This means an improvement by a factor of no less than 50.

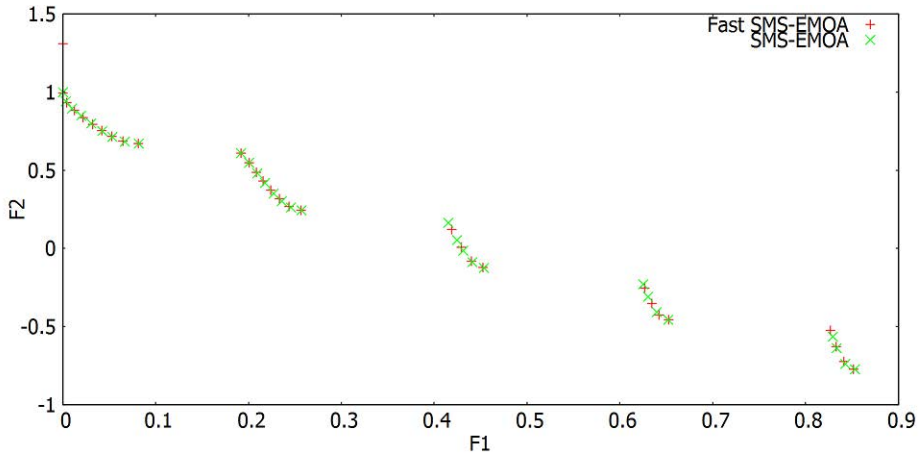

Fig. 9. Result on Zdt3 with SMS-EMOA and Fast SMS-EMOA

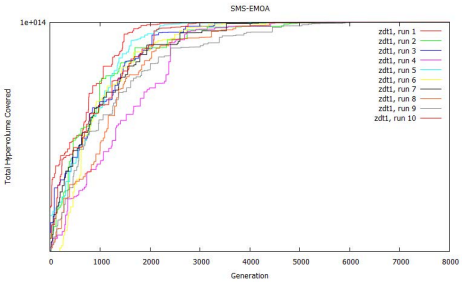

Fig. 10. SMS-EMOA on ZDT1

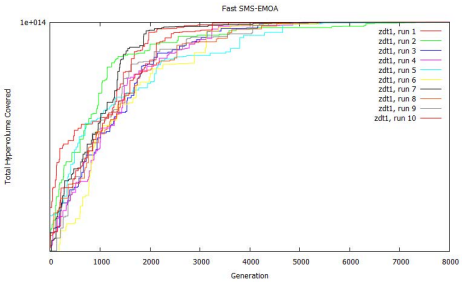

Fig. 11. Fast SMS-EMOA on ZDT1

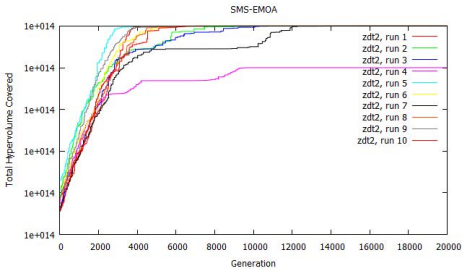

Fig. 12. SMS-EMOA on ZDT2

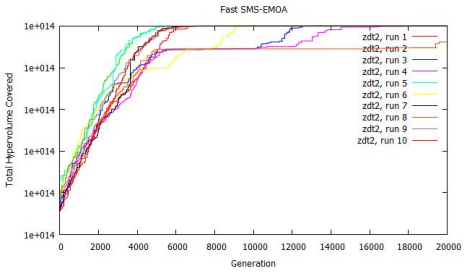

Fig. 13. Fast SMS-EMOA on ZDT2

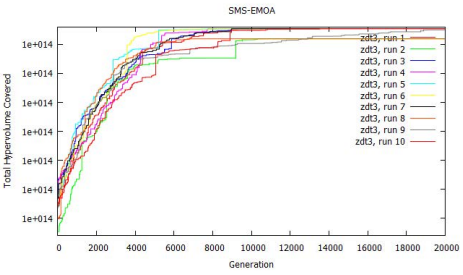

Fig. 14. SMS-EMOA on ZDT3

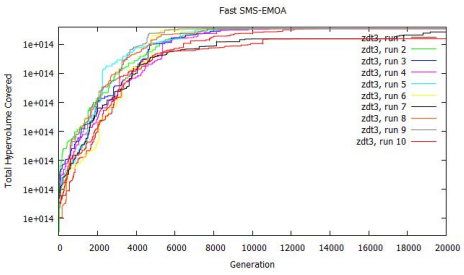

Fig. 15. Fast SMS-EMOA on ZDT3

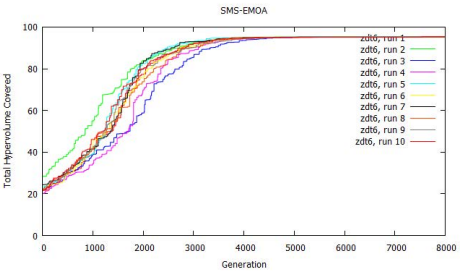

Fig. 16. SMS-EMOA on ZDT6

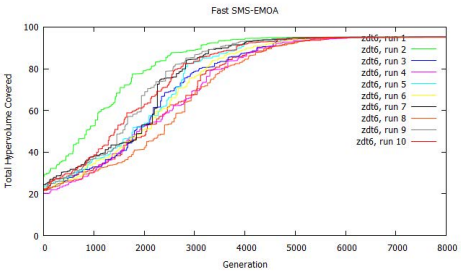

Fig. 17. Fast SMS-EMOA on ZDT6

7 Conclusion and Outlook

Dynamic and incremental updates of 2-D hypervolume-based archives and selection schemes were studied. We showed that maintaining the hypervolume indicator and hypervolume contributions of a bounded-size archive of n points in $\mathbb{R}^2$ can be achieved in asymptotically optimal time $O(\log m)$ per replacement, that is an insertion of a point and subsequent removal of the minimal hypervolume contributor.

For unbounded archives, starting from an empty archive, the hypervolume indicator and all hypervolume contributions can be updated with amortized time complexity in $O(\log n)$ for 2-D. This result extends to 3-D where an amortized time complexity in $O(n)$ can be achieved.

It is discussed how this method can be used to achieve logarithmic time selection schemes for SMS-EMOA and update schemes of bounded-size archiving. For the SMS-EMOA, also an alternative treatment of dominated points using a queue with $O(1)$-time updates was proposed. Empirical tests reveal that especially for large population sizes ($\gg 100$), that would be needed to achieve high-resolution Pareto front approximations, using a fast scheme will be of crucial importance in practice. But even in SMS-EMOA with a population size of only 30 individuals a remarkable acceleration of the selection procedure by a factor of 50 was achieved on standard benchmarks.

Future work will have to deal with a full generalization of results to the higher dimensional case, and include additional benchmarks. Moreover, algorithms similar to SMS-EMOA, e.g. MOO-CMA [13], could benefit from fast update schemes in indicator-based selection, and other alternative selection schemes to non-dominated sorting [15] could be compared.

References

1. Beume, N., Fonseca, C.M., López-Ibáñez, M., Paquete, L., Vahrenhold, J.: On the complexity of computing the hypervolume indicator. IEEE Trans. Evolutionary Computation 13(5), 1075–1082 (2009)
2. Bradstreet, L., Barone, L., While, L.: Updating exclusive hypervolume contributions cheaply. In: Proceedings of the Eleventh Conference on Congress on Evolutionary Computation, CEC 2009, pp. 538–544. IEEE Press, Piscataway (2009)
3. Bringmann, K., Friedrich, T.: Approximating the least hypervolume contributor: NP-hard in general, but fast in practice. In: Ehrgott, M., Fonseca, C.M., Gandibleux, X., Hao, J.-K., Sevaux, M. (eds.) EMO 2009. LNCS, vol. 5467, pp. 6–20. Springer, Heidelberg (2009)
4. Bringmann, K., Friedrich, T.: An efficient algorithm for computing hypervolume contributions. Evol. Comput. 18(3), 383–402 (2010)
5. da Fonseca, V.G., Fonseca, C.M.: The relationship between the covered fraction, completeness and hypervolume indicators. In: Hao, J.-K., Legrand, P., Collet, P., Monmarché, N., Lutton, E., Schoenauer, M. (eds.) EA 2011. LNCS, vol. 7401, pp. 25–36. Springer, Heidelberg (2012)
6. Deb, K.: Multi-objective optimization using evolutionary algorithms. John Wiley & Sons, Hoboken (2001)
7. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multi-objective genetic algorithm, Nsga-ii (2000)
8. Emmerich, M., Beume, N., Naujoks, B.: An EMO algorithm using the hypervolume measure as selection criterion. In: 2005 Intl. Conference, pp. 62–76. Springer (March 2005)
9. Emmerich, M.T.M., Fonseca, C.M.: Computing hypervolume contributions in low dimensions: asymptotically optimal algorithm and complexity results. In: Takahashi, R.H.C., Deb, K., Wanner, E.F., Greco, S. (eds.) EMO 2011. LNCS, vol. 6576, pp. 121–135. Springer, Heidelberg (2011)
10. Fonseca, C.M., Paquete, L., Lopez-Ibanez, M.: An improved dimension-sweep algorithm for the hypervolume indicator, pp. 1157–1163 (July 2006)
11. Landis, E.M., Adelson-Velskii, G.: An algorithm for the organization of information. Proceedings of the USSR Academy of Sciences 146, 263–266
12. Guerreiro, A.P., Fonseca, C.M., Emmerich, M.T.M.: A fast dimension-sweep algorithm for the hypervolume indicator in four dimensions. In: CCCG, pp. 77–82 (2012)
13. Igel, C., Hansen, N., Roth, S.: Covariance matrix adaptation for multi-objective optimization. Evol. Comput. 15(1), 1–28 (2007)
14. Knowles, J.D., Corne, D.W., Fleischer, M.: Bounded archiving using the Lebesgue measure. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 2490–2497. IEEE Press (2003)
15. Naujoks, B., Beume, N., Emmerich, M.T.M.: Multi-objective optimisation using S-metric selection: application to three-dimensional solution spaces, vol. 2, pp. 1282–1289 (September 2005)