A Provably Asymptotically Fast Version of the Generalized Jensen Algorithm for Non-dominated Sorting

Maxim Buzdalov and Anatoly Shalyto

ITMO University
49 Kronverkskiy prosp.
Saint-Petersburg, Russia, 197101
buzdalov@rain.ifmo.ru, shalyto@mail.ifmo.ru

Abstract. The non-dominated sorting algorithm by Jensen, generalized by Fortin et al to handle the cases of equal objective values, has the running time complexity of $O(N \log^{K-1} N)$ in the general case. Here N is the number of points, K is the number of objectives and K is thought to be a constant when N varies. However, the complexity was not proven to be the same in the worst case.

A slightly modified version of the algorithm is presented, for which it is proven that its worst-case running time complexity is $O(N \log^{K-1} N)$.

Keywords: Non-dominated sorting, worst-case running time complexity, multi-objective optimization.

1 Introduction

In the K-dimensional space, a point $A = (a_1, \ldots, a_K)$ is said to *dominate* a point $B = (b_1, \ldots, b_K)$ when for all $1 \le i \le K$ it holds that $a_i \le b_i$ and there exists j such that $a_j < b_j$. *Non-dominated sorting* of points in the K-dimensional space is a procedure of marking all points which are not dominated by any other point with the *rank* of 0, all points which are dominated by at least one point of the rank of 0 are marked with the rank of 1, all points which are dominated by at least one point of the rank $i - 1$ are marked with the rank of i.

Many well-known and widely used multi-objective evolutionary algorithms use the procedure of non-dominated sorting, or the procedure of determining the non-dominated solutions, which can be reduced to non-dominated sorting. These algorithms include NSGA-II [6], PESA [5], PESA-II [4], SPEA2 [11], PAES [9], PDE [2], and many more. The time complexity of a single iteration of these algorithms is often dominated by the complexity of a non-dominated sorting algorithm, so optimization of the latter makes such multi-objective evolutionary algorithms faster.

In Kung et al [10], the algorithm for determining the non-dominated solutions is proposed with the complexity of $O(N \log^{K-1} N)$, where N is the number of points and K is the dimension of the space. It is possible to use this algorithm to

T. Bartz-Beielstein et al. (Eds.): PPSN XIII 2014, LNCS 8672, pp. 528–537, 2014.

perform non-dominated sorting: first, the non-dominated solutions are found and assigned the rank of 0. Then, these solutions are removed, the non-dominated solutions from the remaining ones are found and assigned the rank of 1. The process repeats until all the solutions are removed. This leads to the complexity of $O(N^2 \log^{K-1} N)$ in the worst case, if the maximum rank of a point in the result is $O(N)$.

Jensen [8] was the first to propose an algorithm for non-dominated sorting with the complexity of $O(N \log^{K-1} N)$. However, his algorithm was developed for the assumption that no two points share a common value for any objective, and the complexity was proven for the same assumption. The first attempt to fix this issue belongs, to the best of the authors' knowledge, to Fortin et al [7]. The corrected (or, as in [7], "generalized") algorithm works in all cases, and for the general case the performance is still $O(N \log^{K-1} N)$, but the only upper bound that was proven for the worst case is $O(N^2 K)$.

We present a modified version of this algorithm, for which we prove that its running time is $O(N \log^{K-1} N)$ for fixed K. The rest of the paper is structured as follows. Section 2 contains the description of the new algorithm. In Section 3, the proof of the worst-case running time complexity is given. In Section 4, some words are given on the performance of the original algorithm by Fortin et al. Section 5 concludes.

2 Algorithm Description

In this section, the modified algorithm is described. We try to be as much compatible with the notation of the original paper [7] as possible.

Multi-objective evolutionary algorithms work with candidate solutions to the problem they solve, or *individuals*. Each individual is evaluated and assigned a *fitness*. The fitness $q = f(p)$ of the individual p in a multi-objective problem is a vector of K objectives. We assume that the objectives are to be minimized.

The individuals along with their fitnesses are the input data for the non-dominated sorting algorithm. The algorithm should assign a rank to each individual, as described in Section 1. As two individuals with the same fitness will be assigned the same rank, the algorithm first groups the individuals by their fitnesses, and then works directly with fitnesses, assuming that no two fitnesses are equal in all objectives.

Let q_k be the k-th objective of the fitness q, $q_{1:k}$ be the first k objectives. The relation $q_{1:k} \prec t_{1:k}$ means that q dominates over t in first k objectives. Throughout all the listings in this paper, we denote as $\textsc{rank}(q)$ the rank assigned to the fitness q.

The original algorithm from Fortin et al [7] makes use of the following procedures:

- NonDominatedSort(P, K), the main procedure, receives a population P where each individual has a fitness with K objectives and returns the result of the non-dominated sorting — a sequence of Pareto fronts on which the given individuals reside. It does some preparation work, invokes NDHelperA and constructs the answer.

```
1: procedure NDHelperA(S, k)
2:     if |S| < 2 then return
3:     else if |S| = 2 then
4:         {s^(1), s^(2)} ← S
5:         if s^(1)_{1:k} ≺ s^(2)_{1:k} then
6:             rank(s^(2)) ← max{rank(s^(2)), rank(s^(1)) + 1}
7:         end if
8:     else if k = 2 then
9:         SweepA(S)
10:    else if |{s_k | s ∈ S}| = 1 then
11:        NDHelperA(S, k − 1)
12:    else
13:        L, M, H ← SplitBy(S, median{s_k | s ∈ S}, k)
14:        NDHelperA(L, k)
15:        NDHelperB(L, M, k − 1)
16:        NDHelperA(M, k − 1)
17:        NDHelperB(L ∪ M, H, k − 1)
18:        NDHelperA(H, k)
19:    end if
20: end procedure
```

Fig. 1. The procedure NDHelperA. It assigns ranks to fitnesses from S using the first k objectives.

```
1: procedure NDHelperB(L, H, k)
2:     if L = {} or H = {} then return
3:     else if |L| = 1 or |H| = 1 then
4:         for all h ∈ H, l ∈ L do
5:             if l_{1:k} ⪯ h_{1:k} then
6:                 rank(h) ← max{rank(h), rank(l) + 1}
7:             end if
8:         end for
9:     else if k = 2 then
10:        SweepB(L, H)
11:    else if max{l_k | l ∈ L} ≤ min{h_k | h ∈ H} then
12:        NDHelperB(L, H, k − 1)
13:    else if min{l_k | l ∈ L} ≤ max{h_k | h ∈ H} then
14:        m ← median{s_k | s ∈ L ∪ H}
15:        L_1, M_1, H_1 ← SplitBy(L, m, k)
16:        L_2, M_2, H_2 ← SplitBy(H, m, k)
17:        NDHelperB(L_1, L_2, k)
18:        NDHelperB(L_1, M_2, k − 1)
19:        NDHelperB(M_1, M_2, k − 1)
20:        NDHelperB(L_1 ∪ M_1, M_2, k − 1)
21:        NDHelperB(M_1, M_2, k)
22:    end if
23: end procedure
```

Fig. 2. The procedure NDHelperB. It adjusts ranks of fitnesses from H using the first k objectives by comparing them with fitnesses from L.

```
1: procedure SplitBy(S, m, k)
2:     L ← {s ∈ S|s_k < m}
3:     M ← {s ∈ S|s_k = m}
4:     H ← {s ∈ S|s_k > m}
5:     return L, M, H
6: end procedure
```

Fig. 3. The generic split procedure. In each resulting set, the original order of elements is preserved.

- NDHelperA(S, k) assigns ranks to the fitnesses from the set S based on the first k objectives. It is recursive and may call itself, NDHelperB, SweepA and SplitA.
- NDHelperB(L, H, k) assigns ranks to the fitnesses from the set H by comparing them to the fitnesses from the set L based on the first k objectives. It is recursive and may call itself, SweepB and SplitB.
- SweepA(S) assigns ranks to the fitnesses from the set S based on the first two objectives. It is implemented using the sweep-line approach, and its running time complexity is $O(|S| \log |S|)$.
- SweepB(L, H) assigns ranks to the fitnesses from the set H by comparing them to the fitnesses from the set L based on the first two objectives. It is implemented using the sweep-line approach, and its running time complexity is $O((|L| + |H|) \log |L|)$.
- SplitA(S, k) partitions the set of fitnesses S in two sets around its median for the objective k and balances the resulting sets using the elements equal to the median.
- SplitB(L, H, k) partitions the sets of fitnesses L and H into two sets each around the median of the largest set for the objective k and balances the resulting sets using the elements equal to the median.

Procedures NonDominatedSort, SweepA and SweepB remain the same as in Fortin et al [7]. The procedures SplitA and SplitB are not used by the modified algorithm.

We redefine the procedures NDHelperA, which is shown in Fig. 1, and NDHelperB, which is shown in Fig. 2. These procedures use the procedure SplitBy, which is shown in Fig. 3.

The rest of this section concentrates on differences between the original algorithm by Fortin et al and the proposed algorithm. The correctness of the proposed algorithm can be proven in the same way as the correctness of the original one.

2.1 Splitting into Three Parts, NDHelperA

The difference between the modified NDHelperA and its original version is that the set S is split into three parts (line 12–13) instead of two parts as in Fortin et al [7]. Generally, we have three sets:

– the set L of the elements with the k-th objective less than the median;
– the set M of the elements with the k-th objective equal to the median;
– the set H of the elements with the k-th objective greater than the median.

In the original algorithm, the set M was added either to L or to H, depending on their size. The correctness of the algorithm does not depend on the exact way of splitting. Consider that we split $S = (L \cup M) \cup H$, so that we call:

– NDHelperA($L \cup M$, k);
– NDHelperB($L \cup M$, H, $k - 1$);
– NDHelperA(H, k).

It cannot be predicted without extra assumptions how the set $L \cup M$ will be split in the first recursive call to NDHelperA. However, any split such that all values of the k-th objective in the left are strictly less than all such values on the right is valid, and if the algorithm used it, the correctness would be preserved. In particular, the set $L \cup M$ can be split into L and M. After that, NDHelperA($L \cup M$, k) can be further expanded as follows:

– NDHelperA(L, k);
– NDHelperB(L, M, $k - 1$);
– NDHelperA(M, k).

However, it is known that the k-th objective in M is the same for all fitnesses, which means that NDHelperA(M, k) can be rewritten as NDHelperA(M, $k - 1$). So finally the procedures are called in the following way:

– NDHelperA(L, k);
– NDHelperB(L, M, $k - 1$);
– NDHelperA(M, $k - 1$);
– NDHelperB($L \cup M$, H, $k - 1$);
– NDHelperA(H, k).

This is exactly what one can see in lines 14–18 of the updated version of NDHelperA.

2.2 Splitting into Three Parts, NDHelperB

The ideas from Section 2.1 can be applied to the procedure NDHelperB as well. Assuming we have some pivot (in the original algorithm it is the median of the largest of the L and H sets), we have six sets after the split:

– L_1 — the elements of L with k-th objective less than the pivot;
– M_1 — the elements of L with k-th objective equal to the pivot;
– H_1 — the elements of L with k-th objective greater than the pivot;
– L_2 — the elements of H with k-th objective less than the pivot;
– M_2 — the elements of H with k-th objective equal to the pivot;
– H_2 — the elements of H with k-th objective greater than the pivot.

In the original algorithm, the sets M_1 and M_2 are joined either to L_1 and L_2 or to H_1 and H_2, depending on the sizes of L_1, L_2, H_1, H_2. Note that, although performance depends on the particular choice, correctness does not. To perform the analysis, we merge M_i with L_i. This results in the following sequence of recursive calls:

- NDHelperB($L_1 \cup M_1$, $L_2 \cup M_2$, k);
- NDHelperB($L_1 \cup M_1$, H_2, $k-1$);
- NDHelperB(H_1, H_2, k).

Using the same reasoning as in Section 2.1, the first call can be expanded to:

- NDHelperB(L_1, L_2, k);
- NDHelperB(L_1, M_2, $k-1$);
- NDHelperB(M_1, M_2, k).

The last call can be rewritten as NDHelperB($M_1, M_2, k-1$) because all values of the k-th objective are the same in all fitnesses. So the final call sequence is:

- NDHelperB(L_1, L_2, k);
- NDHelperB(L_1, M_2, $k-1$);
- NDHelperB(M_1, M_2, $k-1$);
- NDHelperB($L_1 \cup M_1$, H_2, $k-1$);
- NDHelperB(H_1, H_2, k).

One can see exactly this sequence in lines 17–21 of the procedure NDHelperB.

2.3 The Choice of Pivot

Consider the last case of NDHelperB where the pivot is chosen. The original choice of Jensen [8], followed by Fortin et al [7], was to choose the pivot in such a way that the sizes of sets in the recursive calls did not exceed $3N/4$, where $N = |H| + |L|$. In the case of the proposed algorithm, it is more important to ensure the sizes of the recursive calls *with the same* k to be small, while not caring for the sizes of the middle sets M_1 and M_2. This forced us to choose the pivot to be the median of k-th objectives of $L \cup H$, which ensures that $|L_1| + |L_2| \leq N/2$ and $|M_1| + |M_2| \leq N/2$.

3 Running Time Estimation

Let us outline the statements about the procedures above:

- In the last case of NDHelperA, $|L| \leq N/2$ and $|H| \leq N/2$, where $N = |S|$. This is due to the choice of the pivot equal to the median of k-th objectives of elements from S.
- In the last case of NDHelperB, $|L_1| + |L_2| \leq N/2$ and $|H_1| + |H_2| \leq N/2$, where $N = |L| + |H|$. As already noted in Section 2.3, this is due to the choice of the pivot.

This helps us to prove the necessary theorems.

3.1 Running Time of NDHelperB

Theorem 1. *The running time of* $\text{NDHelperB}(L, H, k)$ *is*

$$T_B(L, H, k) = O(N \log^{k-1} N)$$

for constant $k \geq 2$*, where* $N = |H| + |L|$.

Proof. Consider the boundary cases where $|L| \leq 1$ or $|H| \leq 1$. In this case, the running time is $\Theta(1)$ or $\Theta(Nk)$, both are $O(N \log N)$.

The non-boundary cases are proven using induction by k. The base case is $k = 2$. $\text{NDHelperB}(L, H, 2)$ just calls $\text{SweepB}(L, H)$, which is $O((|L| + |H|) \log |L|) = O(N \log N)$.

For $k > 2$, assume that the induction statement is proven for $k - 1$. In the worst case, the running time of $\text{NDHelper}(L, H, k)$ is:

$$\begin{aligned} T_B(L, H, k) = {} & T_B(L_1, L_2, k) + \\ & + T_B(L_1, M_2, k-1) + \\ & + T_B(M_1, M_2, k-1) + \\ & + T_B(L_1 \cup M_1, H_2, k-1) + \\ & + T_B(H_1, H_2, k) + \\ & + \Theta(|L| + |H|). \end{aligned}$$

where the $\Theta(|L| + |H|)$ addend corresponds to finding the pivot and splitting the sets. The addends 2–4 and 6 in the expression above can be summarized as $O(N \log^{k-2} N)$ by the induction assumption. It is known that $|L_1| + |L_2| \leq N/2$ and $|H_1| + |H_2| \leq N/2$. If we rewrite $T_B(A, B, k)$ as $T'_B(|A| + |B|, k)$, then:

$$T_B(L, H, k) = T'_B(N, k) \leq 2T'_B(N/2, k) + O(N \log^{k-2} N).$$

From the results of Bentley [3], it follows that:

$$T_B(L, H, k) = O(N \log^{k-1} N). \square$$

3.2 Running Time of NDHelperA

Theorem 2. *The running time of* $\text{NDHelperA}(S, k)$ *is*

$$T_A(S, k) = O(N \log^{k-1} N)$$

for constant $k \geq 2$*, where* $N = |S|$.

Proof. In the boundary case of $|S| \leq 2$, the running time is $\Theta(k) = \Theta(1)$.

The non-boundary cases are proven using induction by k. The base case is $k = 2$. $\text{NDHelperA}(S, 2)$ calls $\text{SweepA}(S)$, which is $O(|S| \log |S|) = O(N \log N)$.

For $k > 2$, assume that the induction statement is proven for $k-1$. The running time of NDHelperA(S,k) is at most:

$$\begin{aligned} T_A(S,k) = {} & T_A(L,k) + \\ & + T_B(L,M,k-1) + \\ & + T_A(M,k-1) + \\ & + T_B(L \cup M,H,k-1) + \\ & + T_A(H,k) + \\ & + \Theta(|S|). \end{aligned}$$

where the $\Theta(|S|)$ addend corresponds to finding the pivot and splitting the sets. The addends 2–4 and 6 can be summarized as $O(N \log^{k-2} N)$ by the induction assumption. It is known that $|L| \leq N/2$ and $|H| \leq N/2$. Using [3], we find that:

$$\begin{aligned} T_A(S,k) = T'_A(N,k) \leq 2T'_A(N/2,k) + O(N \log^{k-2} N) = \\ = O(N \log^{k-1} N). \square \end{aligned}$$

From the last theorem it immediately follows that the running time of the whole algorithm is $O(N \log^{K-1} N)$.

4 Discussion

In this section, two topics are discussed. First, the worst case given in [7] is shown to be not only "very unlikely", but not happening when K is thought to be a constant. Second, some ideas are given that may help to apply the proof above, with some more detailed case analysis, to the original algorithm.

4.1 The Worst-Case Bound for the Original Algorithm

In [7], the worst-case running time estimation for the original algorithm is deduced from the following recurrence:

$$\begin{aligned} T_A(N,K) = {} & T_A(N-1,K) + T_A(1,K) + \\ & + T_B(N-1,1,K-1) + \Theta(N). \end{aligned}$$

Such split can only occur in the case when there is only one element larger than the median. As the median elements are always merged to the set which size is smaller, there is at most one element smaller than the median. We can predict that $T_A(N-1,K)$ will either be split as "one smaller element" and "$N-2$ median elements", or just be delegated to $T_A(N-1,K-1)$.

The actual situation is that, in such heavily imbalanced cases, K decreases as N decreases. This makes the "worst cases" be implementable only if $K = \Theta(N)$. As an example, one may construct the following test case:

$$
\begin{gathered}
(0,0,0,\ldots,0,0,1),\\
(0,0,0,\ldots,0,1,1),\\
\ldots\\
(0,0,1,\ldots,1,1,1),\\
(0,1,1,\ldots,1,1,1),\\
(1,1,1,\ldots,1,1,1).
\end{gathered}
$$

Here the recurrence for the running time for both the original algorithm and the modified version appears to be:

$$
\begin{aligned}
T_A(N,K) = T_A(N-1,K-1) + T_A(1,K) + \\
+ T_B(N-1,1,K-1) + \Theta(N).
\end{aligned}
$$

The solution can be written as $T_A(N,K) = \Theta(N^2K)$. However, $N = K$, so K is not a constant anymore. In the situation of K depending on N, the analysis which produced the expression $O(N \log^{K-1} N)$ actually gives another result, so comparison of the expressions for constant K and for varying K is not valid.

As a final remark to this section, the "proof" of the worst-case performance in the original paper does not prove that there exists a sequence of tests with constant K and growing N, such that the running time of the algorithm grows as $O(N^2K)$. It just considers a class of tests where $K = \Theta(N)$.

4.2 Applicability of the Proof to the Original Algorithm

The authors believe that the proof similar to the one given in this paper can be constructed for the original algorithm from [7]. The reason is that, in the heavily imbalanced cases (when $|M| > \max(|L|,|H|)$ for NDHelperA, and when $|M_1| > \max(|L_1|,|H_1|)$ and $|M_2| > \max(|L_2|,|H_2|)$ for NDHelperB, see Fig. 1 and Fig. 2) the splits in both procedures go in exactly the same way as in the proposed algorithm. As the running time complexity in both well balanced and heavily imbalanced cases is the same, the authors think that the overall complexity is the same.

The immediate application of the proof to the original algorithm cannot be performed because the splits in a partially balanced case for NDHelperB are quite unpredictable due to the algorithm for pivot selection. The proper analysis of such case is welcome.

5 Conclusion

A new version of the non-dominated sorting algorithm is presented. The ideas of the algorithm were first presented by Jensen [8] and corrected by Fortin et al [7] to handle the cases of equal objectives. The presented version differs from the latter in the way the splits in the procedures NDHelperA and NDHelperB are performed, as well as in the way the pivot is selected in NDHelperB.

We proved the running time complexity is $O(N \log^{K-1} N)$ in the worst case for this version.

The implementation of the algorithm, along with some benchmarks, is available at GitHub [1].

This work was financially supported by the Government of Russian Federation, Grant 074-U01.

References

1. Source code for the implementation (a part of this paper), `https://github.com/mbuzdalov/papers/tree/master/2014-ppsn-jensen-fortin`
2. Abbass, H.A., Sarker, R., Newton, C.: PDE: A Pareto Frontier Differential Evolution Approach for Multiobjective Optimization Problems. In: Proceedings of the Congress on Evolutionary Computation, pp. 971–978. IEEE Press (2001)
3. Bentley, J.L.: Multidimensional Divide-and-conquer. Communications of ACM 23(4), 214–229 (1980)
4. Corne, D.W., Jerram, N.R., Knowles, J.D., Oates, M.J.: PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization. In: Proceedings of Genetic and Evolutionary Computation Conference, pp. 283–290. Morgan Kaufmann Publishers (2001)
5. Corne, D.W., Knowles, J.D., Oates, M.J.: The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) PPSN VI. LNCS, vol. 1917, pp. 839–848. Springer, Heidelberg (2000)
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II. Transactions on Evolutionary Computation 6, 182–197 (2000)
7. Fortin, F.A., Grenier, S., Parizeau, M.: Generalizing the Improved Run-time Complexity Algorithm for Non-dominated Sorting. In: Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference, GECCO 2013, pp. 615–622. ACM (2013)
8. Jensen, M.T.: Reducing the Run-time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms. Transactions on Evolutionary Computation 7(5), 503–515 (2003)
9. Knowles, J.D., Corne, D.W.: Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. Evolutionary Computation 8(2), 149–172 (2000)
10. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. Journal of ACM 22(4), 469–476 (1975)
11. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In: Proceedings of the EUROGEN 2001 Conference, pp. 95–100 (2001)