A First Mathematical Runtime Analysis of the Non-dominated Sorting Genetic Algorithm II (NSGA-II)

Weijie Zheng[1], Yufei Liu[2], Benjamin Doerr[2*]

[1] Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China
[2]Laboratoire d'Informatique (LIX), École Polytechnique, CNRS, Institut Polytechnique de Paris, Palaiseau, France
zhengwj13@tsinghua.org.cn, yufei.liu@polytechnique.edu, doerr@lix.polytechnique.fr

Abstract

The non-dominated sorting genetic algorithm II (NSGA-II) is the most intensively used multi-objective evolutionary algorithm (MOEA) in real-world applications. However, in contrast to several simple MOEAs analyzed also via mathematical means, no such study exists for the NSGA-II so far. In this work, we show that mathematical runtime analyses are feasible also for the NSGA-II. As particular results, we prove that with a population size larger than the Pareto front size by a constant factor, the NSGA-II with two classic mutation operators and three different ways to select the parents satisfies the same asymptotic runtime guarantees as the SEMO and GSEMO algorithms on the basic OneMinMax and LOTZ benchmark functions. However, if the population size is only equal to the size of the Pareto front, then the NSGA-II cannot efficiently compute the full Pareto front (for an exponential number of iterations, the population will always miss a constant fraction of the Pareto front). Our experiments confirm the above findings.

Introduction

Most real-world problems contain multiple conflicting objectives. Due to their population-based nature, evolutionary algorithms (EAs) are a natural choice for such problems, and in fact, such multi-objective evolutionary algorithms (MOEAs) have been successfully used in many real-world applications (Zhou et al. 2011).

Unfortunately, the theoretical understanding of MOEAs falls far behind their success in practice. The first mathematical runtime analysis of an MOEA was conducted by Laumanns, Thiele, Zitzler, Welzl, and Deb (2002); Laumanns, Thiele, and Zitzler (2004), who discussed the runtime of the simple evolutionary multi-objective optimizer (SEMO), a bi-objective counterpart of the randomized local search heuristic, on the COCZ and LOTZ benchmarks. Giel (2003) analyzed the global SEMO (GSEMO), the bi-objective counterpart of the $(1+1)$ EA, on the LOTZ function. Subsequent theoretical works majorly focused on variants of these algorithms and analyzed their runtime on the COCZ and LOTZ benchmarks, on variants of them, on new benchmarks, and on combinatorial optimization problems (Qian, Yu, and Zhou 2013; Bian, Qian, and Tang 2018; Roostapour et al. 2019; Qian, Bian, and Feng 2020; Bian et al. 2020; Doerr and Zheng 2021). We note that the (G)SEMO algorithm keeps all non-dominated solutions in the population and discards all others, which can lead to impractically large population sizes. There are two theory works (Brockhoff, Friedrich, and Neumann 2008; Nguyen, Sutton, and Neumann 2015) on the runtime of a simple hypervolume-based EMO algorithm called $(\mu+1)$ SIBEA, regarding both classic benchmarks and problems designed to highlight particular strengths and weaknesses of this algorithm. Like the SEMO and GSEMO, the $(\mu+1)$ SIBEA also creates a single offspring per generation; different from the former, it works with a fixed population size μ. Recently, also decomposition-based multi-objective evolutionary algorithms were analyzed (MOEA/D) (Li et al. 2016; Huang et al. 2019; Huang and Zhou 2020), which decompose the multi-objective problem into several related single-objective problems and then solve each single-objective problem in a co-evolutionary manner. This direction is fundamentally different from the above works and our research. Since it is not primarily focused on multi-objective optimization, we also do not discuss further the successful line of works that solve constrained single-objective problems by turning the constraint violation into a second objective, see, e.g., (Neumann and Wegener 2006; Friedrich et al. 2010; Neumann, Reichel, and Skutella 2011; Friedrich and Neumann 2015; Qian et al. 2017, 2019; Crawford 2021).

As pointed out in the survey (Zhou et al. 2011), the majority of the MOEAs used in research and applications builds on the framework of the non-dominated sorting genetic algorithm II (NSGA-II) (Deb et al. 2002). This algorithm uses a fixed population size N and a complete order defined by the non-dominated sorting and the crowding distance to compare individuals. In each generation, N offspring are generated from the parent population and the N best individuals (according to the complete order) are selected as new parent population. This approach is substantially different from the (G)SEMO algorithm and hypervolume-based approaches (and naturally completely different from decomposition-based methods). Both the predominance in practice and the fundamentally different working principles ask for a rigorous understanding of the NSGA-II, however, to the best of our knowledge so far no mathematical

*Corresponding author.

runtime analysis for the NSGA-II has appeared.[1] We note that the runtime analysis in (Osuna et al. 2020) considers a (G)SEMO algorithm that uses the crowding distance as one of several diversity measures used in the selection of the single parent creating an offspring, but due to the differences of the basic algorithms, none of the arguments used there appears helpful in the analysis of the NSGA-II.

Our contributions. This paper conducts a first step towards theoretically understanding the working principles of the NSGA-II. We regard the NSGA-II with three parent selection strategies (choosing each individual as parent once, choosing parents independently and uniformly at random, and via N independent binary tournaments) and with two classic mutation operators (one-bit mutation and standard bit-wise mutation), but (in this first work) without crossover. We prove that if the population size N is at least a certain constant factor larger than the size of the Pareto front, then each of these six variants of the NSGA-II computes the full Pareto front of the ONEMINMAX benchmark with problem size n in an expected number of $O(n \log n)$ iterations (Theorems 2 and 6) and the front of the LOTZ benchmark in $O(n^2)$ iterations (Theorems 8 and 9). When $N = \Theta(n)$, the corresponding runtime guarantees in terms of fitness evaluations, $O(Nn \log n) = O(n^2 \log n)$ and $O(Nn^2) = O(n^3)$, have the same asymptotic order as those proven previously for the SEMO, GSEMO, and $(\mu + 1)$ SIBEA (when $\mu \geq n + 1$ and $\mu = O(n)$ for the $(\mu + 1)$ SIBEA).

Using a population size larger than the size of the Pareto front is necessary. We prove that if the population size is equal to the size of the Pareto front, then the NSGA-II (applying one-bit mutation once to each parent) regularly loses solutions on the front. This effect is strong enough so that with high probability for an exponential time each generation of the NSGA-II does not cover a constant fraction of the Pareto front of ONEMINMAX. Our experiments confirm these theoretical findings and give some quantitative estimates, e.g., that the fraction of the Pareto front not covered when using population size equal to the front size is around 20% for ONEMINMAX and 40% for LOTZ. Overall, our results show that the NSGA-II despite its higher complexity (parallel generation of offspring, selection based on non-dominated sorting and crowding distance) admits mathematical runtime analyses in a similar fashion as done before for simpler MOEAs.

Preliminaries

In this section, we give a brief introduction to multi-objective optimization and to the NSGA-II framework. For the simplicity of presentation, we shall concentrate on two objectives, both of which have to be maximized. A bi-objective objective function on some search space Ω is a pair $f = (f_1, f_2)$ where $f_i : \Omega \to \mathbb{R}$. We write $f(x) = (f_1(x), f_2(x))$ for all $x \in \Omega$. We shall always assume that we have a bit-string representation, that is, $S = \{0,1\}^n$ for some $n \in \mathbb{N}$. The challenge in multi-objective optimization is that usually there is no solution x that maximizes both f_1 and f_2 and thus is at least as good as all other solutions.

More precisely, in bi-objective maximization, we say x *weakly dominates* y, denoted by $x \succeq y$, if and only if $f_1(x) \geq f_1(y)$ and $f_2(x) \geq f_2(y)$. We say x *strictly dominates* y, denoted by $x \succ y$, if and only if $f_1(x) \geq f_1(y)$ and $f_2(x) \geq f_2(y)$ and at least one of the inequalities is strict. We say that a solution is *Pareto-optimal* if it is not strictly dominated by any other solution. The set of objective values of all Pareto optima is called the *Pareto front* of f. With this language, one aim in multi-objective optimization is to compute a set P of Pareto optima such that $f(P) := \{f(x) \mid x \in P\}$ is the Pareto front. Most previous runtime analyses of MOEAs (and ours) focus on the number of function evaluations used until the MOEA has found such a population; this number is called the *runtime* of the MOEA. We note that in practice, this target often is too ambitious and one therefore restricts oneself to finding a population that approximates well such an ideal solution.

The NSGA-II

When working with a fixed population size, an MOEA must be able to compare any two solutions when selecting the next population. Since dominance is only a partial order, the NSGA-II (Deb et al. 2002) uses the following complete order on the search space. In a given population $P \subseteq \{0,1\}^n$, each individual x has both a rank and a crowding distance. The ranks are defined recursively based on the dominance relation. All individuals that are not strictly dominated by another one have rank one. Given that the ranks $1, \dots, k$ are already defined, the individuals of rank $k+1$ are those among the remaining individuals that are not strictly dominated except by individuals of rank k or smaller. This defines a partition of P into sets $F_1, F_2, \dots$ such that F_i contains all individuals with rank i. It is clear that individuals with lower rank are more interesting, so when comparing two individuals of different ranks, the one with lower rank is preferred.

To compare individuals in the same rank class F_i, the crowding distance of these individuals (in F_i) is computed, and the individual with larger distance is preferred. Ties are broken randomly. Algorithm 1 shows the procedure to calculate the crowding distance in a given set S. The crowding distance of some $x \in S$ is the sum of the crowding distances x has with respect to each objective function f_i. For a given f_i, the individuals in S are sorted in order of ascending f_i value (for equal values, a tie-breaking mechanism is needed, but we shall not make any assumption on this, that is, our mathematical results are valid regardless of how these ties are broken). The first individual and the last individual in the sorted list have an infinite crowding distance. For other individuals in the sorted list, their crowding distance with respect to f_i is the difference of the objective values of its left and right neighbor in the sorted list, normalized by the difference between the first and the last.

The whole NSGA-II framework is shown in Algorithm 2. After the random initialization of the population with size of

[1] We note that by mathematical runtime analysis, we mean the question how many function evaluations a black-box algorithm takes to achieve a certain goal. The computational complexity of the operators used by the NSGA-II, in particular, how to most efficiently implement the non-dominated sorting routine, is a different question (and one that is well-understood (Deb et al. 2002)).

Algorithm 1: crowding-distance(S)

Input: $S = \{S_1, \dots, S_{|S|}\}$: the set of individuals

Output: $\mathrm{cDis}(S) = \big(\mathrm{cDis}\,(S_1), \dots, \mathrm{cDis}\,\big(S_{|S|}\big)\big)$ where $\mathrm{cDis}\,(S_i)$ is the crowding distance for S_i

$\mathrm{cDis}(S) = 0$
for each objective function f_i **do**
  Sort S in order of ascending f_i value: $S_{i.1}, \dots, S_{i.|S|}$
  $\mathrm{cDis}\,(S_{i.1}) = +\infty, \mathrm{cDis}\,\big(S_{i.|S|}\big) = +\infty$
  for $j = 2, \dots, |S| - 1$ **do**
    $\mathrm{cDis}(S_{i.j}) = \mathrm{cDis}(S_{i.j}) + \frac{f_i(S_{i.j+1}) - f_i(S_{i.j-1})}{f_i\big(S_{i.|S|}\big) - f_i(S_{i.1})}$
  end for
end for

Algorithm 2: NSGA-II

Uniformly at random generate the initial population $P_0 = \{x_1, x_2, \dots, x_N\}$ for $x_i \in \{0,1\}^n, i = 1, 2, \dots, N$.
for $t = 0, 1, 2, \dots$ **do**
  Generate the offspring population Q_t with size N
  Using the fast-non-dominated-sort() in (Deb et al. 2002) to divide R_t into $F_1, F_2, \dots$
  Find $i^* > 1$ such that $\sum_{i=1}^{i^*-1} |F_i| < N$ and $\sum_{i=1}^{i^*} |F_i| \geq N$, or $i^* = 1$ for $|F_1| \geq N$
  Using the crowding-distance() in Algorithm 1 to separately calculate the crowding distance of each individual in $F_1, \dots, F_{i^*}$
  Let $\tilde{F}_{i^*}$ be the $N - \sum_{i=0}^{i^*-1} |F_i|$ individuals in F_{i^*} with largest crowding distance, chosen at random in case of a tie
  $P_{t+1} = \left(\cup_{i=1}^{i^*-1} F_i\right) \cup \tilde{F}_{i^*}$
end for

N, the users can apply their way to generate the offspring. Then the total order based on rank and crowding distance is used to remove the worst N individuals in the union of the parent and offspring population. The individuals with smallest rank will survive to the next generation, and the critical rank (that is, if all individuals with this rank survive, then the population size will be at least N but if all individuals do not enter into the next generation, the population size will be less than N) will only have its individuals with largest crowding distances survive to ensure the next population size of N.

Runtime of the NSGA-II on ONEMINMAX

In this section, we analyze the runtime of the NSGA-II on the ONEMINMAX benchmark proposed first by Giel and Lehre (2010) as a bi-objective analogue of the classic ONEMAX benchmark. It is the function $f : \{0,1\}^n \to \mathbb{N} \times \mathbb{N}$ defined by

$$f(x) = \big(f_1(x), f_2(x)\big) = \big(n - \sum_{i=1}^{n} x_i, \sum_{i=1}^{n} x_i\big)$$

for all $x = (x_1, \dots, x_n) \in \{0,1\}^n$. The aim is to maximize both objectives in f. We immediately note that for this benchmark problem, any solution is Pareto optimal and the Pareto front is $\{(0,n), (1,n-1), \dots, (n,0)\}$. It is hence a good example to study how an EMO algorithm explores the Pareto front when already some Pareto optima were found.

Giel and Lehre (2010) showed that the simple SEMO algorithm finds the full Pareto front of ONEMINMAX in $O(n^2 \log n)$ iterations and fitness evaluations. Their proof can easily be extended to the GSEMO algorithm. For the SEMO, a (matching) lower bound of $\Omega(n^2 \log n)$ was shown by Osuna, Gao, Neumann, and Sudholt (2020). An upper bound of $O(\mu n \log n)$ was shown for the hypervolume-based $(\mu+1)$ SIBEA with $\mu \geq n+1$ (Nguyen, Sutton, and Neumann 2015). When the SEMO or GSEMO is enriched with a diversity mechanism (strong enough so that solutions that can create a new point on the Pareto front are chosen with constant probability), then the runtime of these algorithms reduces to $O(n \log n)$.

In contrast to the SEMO and GSEMO as well as the $(\mu+1)$ SIBEA with population size $\mu \geq n+1$, the NSGA-II can lose all solutions covering a point of the Pareto front. In the following lemma, central to our runtime analyses on ONEMINMAX, we show that this cannot happen when the population size is large enough, namely at least four times the size of the Pareto front.

Lemma 1. *Consider one iteration of the NSGA-II with population size $N \geq 4(n+1)$ optimizing the* ONEMINMAX *function. Assume that in some iteration t the combined parent and offspring population $R_t = P_t \cup Q_t$ contains a solution x with objective value $(k, n-k)$ for some $k \in [0..n]$. Then also the next parent population P_{t+1} contains an individual y with $f(y) = (k, n-k)$.*

Since Lemma 1 ensures that objective values on the Pareto front will not be lost in the future, we can estimate the runtime of the NSGA-II via the sum of the waiting times for finding a new Pareto solution. Apart from the fact that the NSGA-II generates N solutions per iteration (which requires some non-trivial arguments in the case of binary tournament selection), this analysis resembles the known analysis of the simpler SEMO algorithm (Giel and Lehre 2010). For $N = O(n)$, we also obtain the same runtime estimate.

We start with the easier case that parents are chosen uniformly at random or that each parent creates one offspring.

Theorem 2. *Consider optimizing the* ONEMINMAX *function via the NSGA-II with one of the following four ways to generate the offspring population in Step 3 in Algorithm 2, namely applying one-bit mutation or standard bit-wise mutation once to each parent or N times choosing a parent uniformly at random and applying one-bit mutation or standard bit-wise mutation to it. If the population size N is at least $4(n+1)$, then the expected runtime is $O(n \log n)$ iterations and $O(Nn \log n)$ fitness evaluations.*

We now analyze the performance of NSGA-II on ONEMINMAX when selecting the parents via binary tournaments, which is selection method suggested in the original NSGA-II paper (Deb et al. 2002). Here the offspring population Q_t is generated by N times independently performing the following sequence of actions: (i) Select two different individuals x', x'' uniformly at random from P_t. (ii) Select x

as the better of these two, that is, the one with smaller rank in P_t or, in case of equality, the one with larger crowding distance in P_t (breaking a tie randomly). [2] (iii) Generate an offspring by mutating x.

In the case of binary tournament selection, the analysis is slightly more involved since we need to argue that a desired parent is chosen for mutation with constant probability in one iteration. This is easy to see for a parent at the boundary of the front as its crowding distance is infinite, but less obvious for parents not at the boundary. We note that we need to be able to select such parents since we cannot ensure that the population intersects the Pareto front in a contiguous interval (as can be seen, e.g., from the random initial population). We solve this difficulty in the following three lemmas.

We use the following notation. Consider some iteration t. For $i = 1, 2$, let

$$v_i^{\min} = \min\{f_i(x) \mid x \in R_t\},$$
$$v_i^{\max} = \max\{f_i(x) \mid x \in R_t\}$$

denote the extremal objective values. Let $V = f(R_t) = \{(f_1(x), f_2(x)) \mid x \in R_t\}$ denote the set of objective values of the solutions in the combined parent and offspring population R_t. We define the set of values such that also the right (left) neighbor on the Pareto front is covered by

$$V_{\text{in}}^+ = \{(v_1, v_2) \in V \mid \exists y \in R_t : (f_1(y), f_2(y)) = (v_1 + 1, v_2 - 1)\},$$
$$V_{\text{in}}^- = \{(v_1, v_2) \in V \mid \exists y \in R_t : (f_1(y), f_2(y)) = (v_1 - 1, v_2 + 1)\}.$$

Lemma 3. *For any $(v_1, v_2) \in V \setminus (V_{\text{in}}^+ \cap V_{\text{in}}^-)$, there is at least one individual $x \in R_t$ with $f(x) = (v_1, v_2)$ and $\text{cDis}(x) \geq \frac{2}{v_1^{\max} - v_1^{\min}}$.*

Lemma 4. *For any $(v_1, v_2) \in V_{\text{in}}^+ \cap V_{\text{in}}^-$, there are at most two individuals in R_t with objective value (v_1, v_2) and crowding distance at least $\frac{2}{v_1^{\max} - v_1^{\min}}$.*

Lemma 5. *Assume that R_t does not cover the whole Pareto front. Let R_t^N be the set of individuals that are neighbors of an uncovered point on the Pareto front, that is $R_t^N := \{x \in R_t \mid \exists \epsilon \in \{-1, +1\} : (f_1(x) + \epsilon, f_2(x) - \epsilon)) \in [0..n]^2 \setminus f(R_t)\}$. With probability at least $\frac{1}{2N} \frac{N-10}{N-1}$, the winner of a binary tournament lies in R_t^N.*

With Lemma 5, we can now easily argue that in a given iteration t, we have a constant probability of choosing at least once a parent that is a neighbor of an empty spot on the Pareto front. This allows to re-use the main arguments of the simpler analyses for the cases that the parents were choosing randomly or that each parent creates one offspring.

Theorem 6. *Consider optimizing the* OneMinMax *function via the NSGA-II which creates the offspring population by N times choosing a parent via binary tournament selection and applying one-bit or standard bit-wise mutation to it. If the population size N is at least $4(n + 1)$, then the expected runtime is $O(n \log n)$ iterations and $O(Nn \log n)$ fitness evaluations.*

[2] Although the strategy is first domination criterium between two solutions and then crowding distance for the binary tournament selection in Deb's original C code, the NSGA-II in the recent pymoo platform (https://pymoo.org/algorithms/moo/nsga2.html) by Blank and Deb utilizes this first rank then crowding distance strategy. We adhere to the recent strategy.

Runtime of the NSGA-II on LOTZ

We proceed with analyzing the runtime of the NSGA-II on the benchmark LOTZ proposed by Laumanns, Thiele, and Zitzler (2004). This is the function $f : \{0, 1\}^n \to \mathbb{N} \times \mathbb{N}$ defined by

$$f(x) = \big(f_1(x), f_2(x)\big) = \Big(\sum_{i=1}^{n} \prod_{j=1}^{i} x_j, \sum_{i=1}^{n} \prod_{j=i}^{n} (1 - x_j)\Big)$$

for all $x \in \{0, 1\}^n$. Here the first objective is the so-called LeadingOnes function, counting the number of (contiguous) leading ones of the bit string, and the second objective counts in an analogous fashion the number of trailing zeros. Again, the aim is to maximize both objectives. Different from OneMinMax, here many solutions exist that are not Pareto optimal, in fact, only $0^n, 10^{n-1}, \ldots, 1^n$ are the Pareto optima of this problem. The Pareto front is again $\{(0, n), (1, n-1), \ldots, (n, 0)\}$. The known runtimes for this benchmark are $\Theta(n^3)$ for the SEMO (Laumanns, Thiele, and Zitzler 2004), an upper bound of $O(n^3)$ for the GSEMO with standard mutation rate $1/n$ (Giel 2003) and a lower bound of $\Omega(n^2/p)$ for mutation rate $p \leq n^{7/4}$ (Doerr, Kodric, and Voigt 2013), and an upper bound of $O(\mu n^2)$ for the $(\mu + 1)$ SIBEA with population size $\mu \geq n + 1$ (Brockhoff, Friedrich, and Neumann 2008).

Similar to OneMinMax, we can show that when the population size is large enough, an objective value on the Pareto front stays in the population from the point on when it is discovered.

Lemma 7. *Consider one iteration of the NSGA-II with population size $N \geq 4(n + 1)$ optimizing the* LOTZ *function. Assume that in some iteration t the combined parent and offspring population $R_t = P_t \cup Q_t$ contains a solution x with rank one. Then also the next parent population P_{t+1} contains an individual y with $f(y) = f(x)$. In particular, once the parent population contains an individual with objective value $(k, n - k)$, it will do so for all future generations.*

Since not all individuals are on the Pareto front, the runtime analysis for LOTZ function is slightly more complex than for OneMinMax. The process contains two stages: the first stage lasts until we have a solution on the Pareto front. In this phase, we argue that the first objective value increases by one every (expected) $O(n)$ iterations. Consequently, after an expected number of $O(n^2)$ iterations, we have an individual x in the population with $f_1(x) = n$, which necessarily lies on the Pareto front. The second stage, where we complete the Pareto front from an existing Pareto-solution, can be analyzed in a similar manner as for OneMinMax in Theorem 2, noting of course the different probabilities to generate a new solution on the Pareto front.

Theorem 8. *Consider optimizing the* LOTZ *function via the NSGA-II with one of the following four ways to generate the*

offspring population in Step 3 in Algorithm 2, namely applying one-bit mutation or standard bit-wise mutation once to each parent or N times choosing a parent uniformly at random and applying one-bit mutation or standard bit-wise mutation to it. If the population size N is at least $4(n+1)$, then the expected runtime is $O(n^2)$ iterations and $O(Nn^2)$ fitness evaluations.

We now study the runtime of NSGA-II using binary tournament selection. Compared to ONEMINMAX, we face the additional difficulty that now rank one solutions can exist which are not on the Pareto front. Due to their low rank, they could perform well in the selection, but being possibly far from the front, they are not interesting as parents. To ensure that desired parents have a sufficient chance of winning the tournaments, we increase our lower bound on the population size to $5(n+1)$. We note that we did not try hard to find the smallest population size allowing to prove our result, so possibly the following theorem could also be shown with a population size of $4(n+1)$, but definitely this would not be possible with the arguments used in the case of ONEMINMAX.

Theorem 9. *Consider optimizing the* LOTZ *function via the NSGA-II with N times choosing a parent by binary tournament selection and applying one-bit or standard bit-wise mutation to it. If the population size N is at least $5(n+1)$, then the expected runtime is $O(n^2)$ iterations and $O(Nn^2)$ fitness evaluations.*

An Exponential Lower Bound for Small Population Size

In this section, we prove a lower bound for a small population size. Since lower bound proofs can be quite complicated, recall for example that there are matching lower bounds for the runtime of the SEMO (using one-bit mutation) on ONEMINMAX and LOTZ, but not for the GSEMO (using bit-wise mutation), we restrict ourselves to the simplest variant using each parent once to generate one offspring via one-bit mutation. From the proofs, though we are optimistic that our results, possibly with different implicit constants, can also be shown for all other variants of the NSGA-II regarded in this work.

Our main result is that this NSGA-II takes an exponential time to find the whole Pareto front (of size $n+1$) of ONEMINMAX when the population size is $n+1$. This is different from the SEMO and GSEMO algorithms (which have no fixed population size, but which will never store a population larger than $n+1$ when optimizing ONEMINMAX) and the $(\mu+1)$ SIBEA with population size $\mu = n+1$. Even stronger, we show that there is a constant $\varepsilon > 0$ such that when the current population P_t covers at least $|f(P_t)| \geq (1-\varepsilon)(n+1)$ points on the Pareto front of ONEMINMAX, then with probability $1-\exp(-\Theta(n))$, the next population P_{t+1} will cover at most $|f(P_{t+1})| \leq (1-\varepsilon)(n+1)$ points on the front. Hence when a population covers a large fraction of the Pareto front, then with very high probability the next population will cover fewer points on the front. When the coverage is smaller, that is, $|f(P_t)| \leq (1-\varepsilon)(n+1)$, then with probability $1-\exp(-\Theta(n))$ the combined parent and offspring population R_t will miss a constant fraction of the Pareto front. From these two statements, it is easy to see that there is a constant δ such that with probability $1-\exp(-\Omega(n))$, in none of the first $\exp(\Omega(n))$ iterations the combined parent and offspring population covers more than $|f(R_t)| \geq (1-\delta)(n+1)$ points of the Pareto front.

Let us discuss the two proof arguments in detail, starting with the latter one. When the current population is still missing a constant fraction of the Pareto front, then the following arguments show that with probability $1-\exp(-\Theta(n))$, also the combined parent and offspring population R_t (and thus also the next population $P_{t+1} \subseteq R_t$) will miss a constant fraction of the front. With a constant fraction of the front missing in P_t, also a constant fraction that is $\Omega(n)$ away from the boundary points $(0,n)$ and $(n,0)$ is missing. These values have the property that from both their neighboring positions, they can only be generated with constant probability via one-bit mutation. Again a constant fraction of these have only a constant number of individuals on neighboring positions. These, thus have a (small) constant probability of not being generated in this iteration. This shows that in expectation, we are missing a constant fraction of the Pareto front. Via the method of bounded differences (exploiting that each mutation operation can change the number of missing elements by at most one), we turn this expectation into a bound that holds with probability $1-\exp(-\Omega(n))$.

We now turn to the other argument, which is that when the current population covers the Pareto front to a large extent, then the selection procedure of the NSGA-II will increase the number of uncovered points so that at least $\epsilon(n+1)$ positions are uncovered by P_{t+1}. The key arguments to show this claim are the following. When a large part of the front is covered, then many points can be only covered by a single individual of P_t (since the population size equals the size of the front). With some careful counting, we derive from this that (by a constant factor) more than half of the positions on the front are covered exactly twice in the combined parent and offspring population R_t. Almost all of them have both their neighboring positions covered as well. This shows that (by a constant factor) more than $(n+1)$ of the individuals in R_t have the same crowding distance. Consequently, within this set of individuals the selection operator takes a random choice, which in expectation will remove both individuals from a constant fraction of the points on the Pareto front. Again, the method of bounded differences turns this expectation into a statement with probability $1-\exp(-\Omega(n))$.

We have the formal results in the following.

Theorem 10. *Consider optimizing* ONEMINMAX *via the NSGA-II applying one-bit mutation once to each individual. Let the population size be $N = n+1$. Then the expected number of iterations until the population covers the full Pareto front is* $\exp(\Omega(n))$.

Experiments

To complement our asymptotic results with data for concrete problem sizes, we conducted the following experiments.

Settings

We use, in principle, the version of the NSGA-II given by Deb (Revision 1.1.6), available at https://www.egr.msu.edu/~kdeb/codes.shtml, except that, as in our theoretical analysis, we do not use crossover. We re-implemented the algorithm in Matlab (R2016b). When a sorting procedure is used, we use the one provided by Matlab (and not randomized Quicksort as in Deb's implementation). As mentioned before, we utilize the first rank then crowding distance strategy for the binary tournament selection as in the recent pymoo platform (https://pymoo.org/algorithms/moo/nsga2.html) by Blank and Deb.

Our theoretical analysis above covers three parent selection strategies and two mutation operators. For reasons of space, we concentrate our experiments on one representative setting, namely binary tournament selection (as proposed in (Deb et al. 2002)) and standard bit-wise mutation with mutation rate $\frac{1}{n}$ (which is the most common mutation operator in evolutionary computation). We use the following experimental settings.

- Problem size n: $100, 200, 300,$ and 400 for OneMinMax, and $30, 60, 90,$ and 120 for LOTZ.
- Population size N: Our theoretical analyses (Theorems 6 and 9) proved a good performance of the NSGA-II for population sizes of at least $N^* = 4(n+1)$ for OneMinMax and at least $N^* = 5(n+1)$ for LOTZ. We use this value also in the experiments. We also use the value $N = 2N^*$, for which our theory results apply, but our runtime guarantees are twice as large as for N^* (when making the implicit constants in the results visible). We also use two smaller values (of at least $1.5(n+1)$) to explore the parameter range where we have no proven result, but where one can be optimistic that roughly the same working principles apply and the smaller population size possibly gives a better performance. Finally, we conduct experiments with the population size $N = n+1$, which is large enough to represent the full Pareto front, but for which we have proven the NSGA-II to be ineffective (on OneMinMax and when letting each parent create an offspring via one-bit mutation).

Efficient Population Sizes

Figure 1 displays the runtime (that is, the time until the full Pareto front is covered) of the NSGA-II with the four "large" population sizes together with the runtime of the (parameterless) GSEMO. This data confirms that the NSGA-II can efficiently cover the Pareto front of OneMinMax and LOTZ when using a population size of at least N^*. The runtimes for $N = 2N^*$ are clearly larger than for N^*, roughly by a factor of 2 for LOTZ, but by slightly less for OneMinMax. The data for the two population sizes smaller than N^* indicates that also for these parameter settings the NSGA-II performs very well.

Comparing the NSGA-II to the GSEMO, we observe that the NSGA-II with a proper choice of the population size shows a better performance. This is interesting and somewhat unexpected (on simple problems like OneMinMax and LOTZ) since the NSGA-II throughout the run works with a population of size N, whereas the GSEMO only keeps the at most $n+1$ non-dominated individuals in its population. Consequently, in particular in the early stages of the optimization process, each iteration takes significantly fewer fitness evaluations. It is clear that the NSGA-II with binary tournament selection has the advantage of favoring extremal individuals as parents, but that this advantage outnumbers the disadvantage of the higher cost per iteration – at least a factor of two when using $N = 2(n+1)$ – was not expected by us.

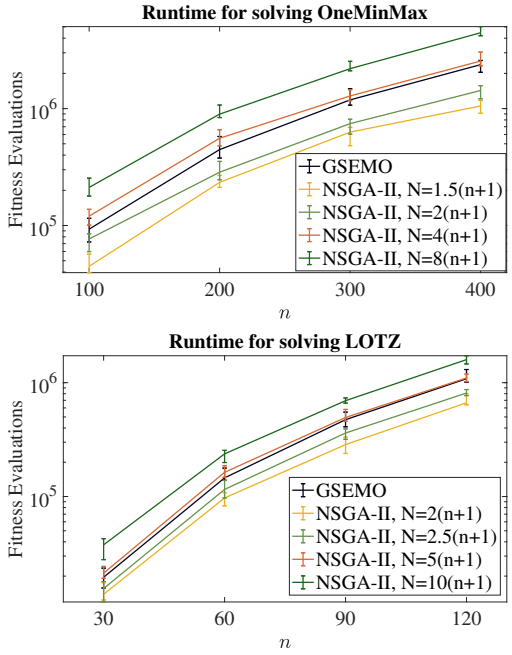

Figure 1: The number of function evaluations for the NSGA-II (binary tournament selection, standard bit-wise mutation) with different population sizes and for the GSEMO optimizing OneMinMax and LOTZ. Displayed are the median (with 1st and 3rd quartiles) in 20 independent runs.

Inefficient Population Size

When the population size is small, we do not have the result that points on the front cannot be lost (Lemmas 1 and 7) and the proof of Theorem 10 shows that indeed we can easily lose points on the front, leading to a runtime at least exponential in n when $N = n+1$. In this subsection, we analyze this phenomenon experimentally. As discussed earlier, we concentrate on the NSGA-II with tournament selection and standard bit-wise mutation (but we have no reason to believe that both the mathematical and the experimental findings hold for all variants of the NSGA-II).

Since it is hard to show experimentally that a runtime is at least exponential, we do not run the algorithm until it found the full Pareto front (this would be possible only for very small problem sizes), but we conduct a slightly different experiment for reasonable problem sizes which also strongly indicates that the NSGA-II has enormous difficulties in find-

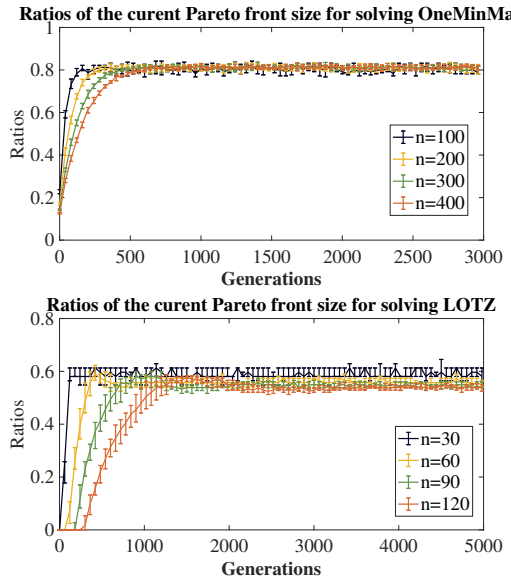

Figure 2: Ratio of the coverage of the Pareto front by the current population of the NSGA-II (binary tournament selection, standard bit-wise mutation) with population size $N = n + 1$. Displayed are the median (with 1st and 3rd quartiles) in 20 independent runs.

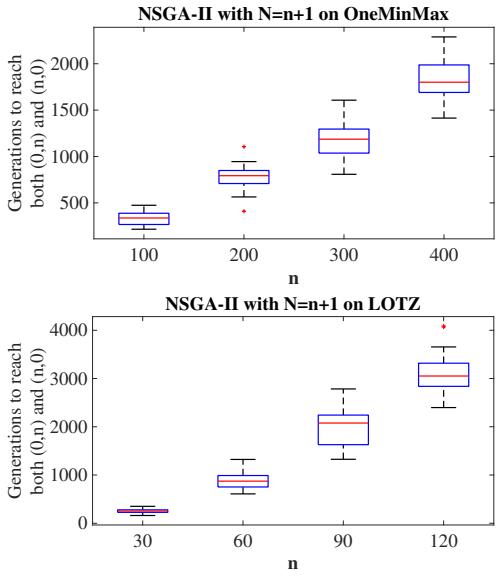

Figure 3: First generation when both extreme function values $(0, n)$ and $(n, 0)$ were contained in the population of the NSGA-II (binary tournament selection, standard bit-wise mutation, population size $N = n + 1$). Displayed are the median (with 1st and 3rd quartiles) in 20 independent runs.

ing the full front. We ran the NSGA-II for 3000 generations for ONEMINMAX and 5000 generations for LOTZ and measured for each generation the ratio by which the Pareto front is covered. This data is displayed in Figure 2. We see clearly that the coverage of the Pareto front steeply increases at first, but then stagnates at a constant fraction clearly below one (around 80% for ONEMINMAX and between 50% and 60% for LOTZ) and this in a very concentrated manner. From this data, there is no indication that the Pareto front will be covered anytime soon. Back-of-the-envelope calculations (similar to the rigorous proof of Theorem 10) suggest that, regardless of the variant of the algorithm we consider, when P_t covers only a constant fraction of the front, then the probability that R_t covers the full front is $\exp(-\Omega(n))$. The main argument, as in the proof of Theorem 10, would be that a constant fraction of the missing points on the front has only a constant probability (less than one) of being generated. Hence in expectation, a constant fraction will not be generated, and the method of bounded differences (exploiting the large amount of independent randomness in the offspring generation) translates this expectation into a statement that holds with probability $1 - \exp(-\Omega(n))$.

Without exploring this aspect in detail, we note that the NSGA-II, while not finding the full Pareto front, still approximates the front very well. Figure 3 shows that also with this low population size, the NSGA-II discovers the extremal points $(0, n)$ and $(n, 0)$ of the front relatively fast. When regarding the solutions on the Pareto front after 3000 generations (ONEMINMAX) and 5000 generations (LOTZ), on all runs we never encountered an interval of uncovered points of length longer than 4.

Conclusion and Future Work

This paper conducted the first mathematical runtime analysis of the NSGA-II, which is the predominant framework in real-world multi-objective optimization. We proved that with a suitable population size, several variants of the NSGA-II satisfy the same runtime guarantees as the SEMO, GSEMO, and $(\mu + 1)$ SIBEA when optimizing the two benchmarks ONEMINMAX and LOTZ. The choice of the population size is important. We prove an exponential runtime of the NSGA-II when the population size equals the size of the Pareto front.

On the technical side, this paper shows that mathematical runtime analyses are feasible also for the NSGA-II. We provided a number of arguments to cope with the specific properties of this algorithm, in particular, the fact that points in the Pareto front can be lost and the commonly used parent selection via binary tournaments based on the rank and crowding distance. We are optimistic that these tools will aid future analyses of the NSGA-II.

In this first analysis of the NSGA-II, following the example of most previous theoretical works on MOEAs, we only regarded the target of computing the full Pareto front. For many practically relevant problems, this is too much to ask for since the Pareto front can be very large. In this case, a better target is to compute a smaller set of Pareto optima that is a good sparse representation of the full front. One of our experiments for the case $N = n + 1$ suggests that the NSGA-II could be well suited to compute such approximations. Understanding this research question will be the target of our future research.

Acknowledgments

This work was supported in part by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH, by Guangdong Basic and Applied Basic Research Foundation (Grant No. 2019A1515110177), by Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), by the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), and by Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531).

References

Bian, C.; Feng, C.; Qian, C.; and Yu, Y. 2020. An efficient evolutionary algorithm for subset selection with general cost constraints. In *AAAI Conference on Artificial Intelligence, AAAI 2020*, 3267–3274. AAAI.

Bian, C.; Qian, C.; and Tang, K. 2018. A general approach to running time analysis of multi-objective evolutionary algorithms. In *International Joint Conference on Artificial Intelligence, IJCAI 2018*, 1405–1411. IJCAI.

Brockhoff, D.; Friedrich, T.; and Neumann, F. 2008. Analyzing hypervolume indicator based algorithms. In *Parallel Problem Solving from Nature, PPSN 2008*, 651–660. Springer.

Crawford, V. G. 2021. Faster guarantees of evolutionary algorithms for maximization of monotone submodular functions. In *International Joint Conference on Artificial Intelligence, IJCAI 2021*, 1661–1667. ijcai.org.

Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6: 182–197.

Doerr, B.; Kodric, B.; and Voigt, M. 2013. Lower bounds for the runtime of a global multi-objective evolutionary algorithm. In *IEEE Congress on Evolutionary Computation, CEC 2013*, 432–439. IEEE.

Doerr, B.; and Zheng, W. 2021. Theoretical analyses of multi-objective evolutionary algorithms on multi-modal objectives. In *AAAI Conference on Artificial Intelligence, AAAI 2021*, 12293–12301. AAAI.

Friedrich, T.; He, J.; Hebbinghaus, N.; Neumann, F.; and Witt, C. 2010. Approximating covering problems by randomized search heuristics using multi-objective Models. *Evolutionary Computation*, 18: 617–633.

Friedrich, T.; and Neumann, F. 2015. Maximizing submodular functions under matroid constraints by evolutionary algorithms. *Evolutionary Computation*, 23: 543–558.

Giel, O. 2003. Expected runtimes of a simple multi-objective evolutionary algorithm. In *IEEE Congress on Evolutionary Computation, CEC 2003*, volume 3, 1918–1925. IEEE.

Giel, O.; and Lehre, P. K. 2010. On the effect of populations in evolutionary multi-objective optimisation. *Evolutionary Computation*, 18: 335–356.

Huang, Z.; and Zhou, Y. 2020. Runtime analysis of somatic contiguous hypermutation operators in MOEA/D framework. In *AAAI Conference on Artificial Intelligence, AAAI 2020*, volume 34, 2359–2366. AAAI.

Huang, Z.; Zhou, Y.; Chen, Z.; and He, X. 2019. Running time analysis of MOEA/D with crossover on discrete optimization problem. In *AAAI Conference on Artificial Intelligence, AAAI 2019*, volume 33, 2296–2303. AAAI.

Laumanns, M.; Thiele, L.; and Zitzler, E. 2004. Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. *IEEE Transactions on Evolutionary Computation*, 8: 170–182.

Laumanns, M.; Thiele, L.; Zitzler, E.; Welzl, E.; and Deb, K. 2002. Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem. In *International Conference on Parallel Problem Solving from Nature, PPSN 2002*, 44–53. Springer.

Li, Y.-L.; Zhou, Y.-R.; Zhan, Z.-H.; and Zhang, J. 2016. A primary theoretical study on decomposition-based multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 20: 563–576.

Neumann, F.; Reichel, J.; and Skutella, M. 2011. Computing minimum cuts by randomized search heuristics. *Algorithmica*, 59: 323–342.

Neumann, F.; and Wegener, I. 2006. Minimum spanning trees made easier via multi-objective optimization. *Natural Computing*, 5: 305–319.

Nguyen, A. Q.; Sutton, A. M.; and Neumann, F. 2015. Population size matters: Rigorous runtime results for maximizing the hypervolume indicator. *Theoretical Computer Science*, 561: 24–36.

Osuna, E. C.; Gao, W.; Neumann, F.; and Sudholt, D. 2020. Design and analysis of diversity-based parent selection schemes for speeding up evolutionary multi-objective optimisation. *Theoretical Computer Science*, 832: 123–142.

Qian, C.; Bian, C.; and Feng, C. 2020. Subset selection by Pareto optimization with recombination. In *AAAI Conference on Artificial Intelligence, AAAI 2020*, 2408–2415. AAAI.

Qian, C.; Shi, J.-C.; Yu, Y.; and Tang, K. 2017. On subset selection with general cost constraints. In *International Joint Conference on Artificial Intelligence, IJCAI 2017*, volume 17, 2613–2619. ijcai.org.

Qian, C.; Yu, Y.; Tang, K.; Yao, X.; and Zhou, Z. 2019. Maximizing submodular or monotone approximately submodular functions by multi-objective evolutionary algorithms. *Artificial Intelligence*, 275: 279–294.

Qian, C.; Yu, Y.; and Zhou, Z.-H. 2013. An analysis on recombination in multi-objective evolutionary optimization. *Artificial Intelligence*, 204: 99–119.

Roostapour, V.; Neumann, A.; Neumann, F.; and Friedrich, T. 2019. Pareto optimization for subset selection with dynamic cost constraints. In *AAAI Conference on Artificial Intelligence, AAAI 2019*, volume 33, 2354–2361. AAAI.

Zhou, A.; Qu, B.-Y.; Li, H.; Zhao, S.-Z.; Suganthan, P. N.; and Zhang, Q. 2011. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1: 32–49.