



**HAL**  
open science

# A First Runtime Analysis of the NSGA-II on a Multimodal Problem

Benjamin Doerr, Zhongdi Qu

► **To cite this version:**

Benjamin Doerr, Zhongdi Qu. A First Runtime Analysis of the NSGA-II on a Multimodal Problem. Parallel Problem Solving from Nature (PPSN 2022), Sep 2022, Dortmund, Germany. 10.1007/978-3-031-14721-0\_28 . hal-03797602

**HAL Id: hal-03797602**

**<https://hal.science/hal-03797602>**

Submitted on 4 Oct 2022

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# A First Runtime Analysis of the NSGA-II on a Multimodal Problem

Benjamin Doerr<sup>1</sup> and Zhongdi Qu<sup>1</sup>

Laboratoire d'Informatique (LIX), Ecole Polytechnique, CNRS, Institut Polytechnique de Paris, Palaiseau, France

**Abstract.** Very recently, the first mathematical runtime analyses of the multi-objective evolutionary optimizer NSGA-II have been conducted (AAAI 2022, GECCO 2022 (to appear), arxiv 2022). We continue this line of research with a first runtime analysis of this algorithm on a benchmark problem consisting of two multimodal objectives. We prove that if the population size  $N$  is at least four times the size of the Pareto front, then the NSGA-II with four different ways to select parents and bit-wise mutation optimizes the OneJumpZeroJump benchmark with jump size  $2 \leq k \leq n/4$  in time  $O(Nn^k)$ . When using fast mutation, a recently proposed heavy-tailed mutation operator, this guarantee improves by a factor of  $k^{\Omega(k)}$ . Overall, this work shows that the NSGA-II copes with the local optima of the OneJumpZeroJump problem at least as well as the global SEMO algorithm.

**Keywords:** NSGA-II · Multimodal Problem · Runtime Analysis.

## 1 Introduction

The mathematical runtime analysis of evolutionary algorithms (EAs) has contributed significantly to our understanding of these algorithms, has given advice on how to set their parameters, and has even proposed new algorithms [10,1,9,7]. Most of the insights, however, have been obtained by regarding artificially simple algorithms such as the  $(1+1)$  EA, the fruit fly of EA research.

In contrast, the recent work [13] succeeded in analyzing the *non-dominated sorting genetic algorithm II (NSGA-II)* [5], the multi-objective EA (MOEA) most used in practice [14]. This line of research was almost immediately followed up in [2] and [12]. These three works, just like the majority of the theoretical works on MOEAs, only regard multi-objective problems composed of unimodal objectives (see Section 2 for more details).

In this work, we continue the runtime analysis of the NSGA-II with a first analysis on a problem composed of two multi-modal objectives, namely the ONEJUMPZEROJUMP problem proposed in [8]. This problem, defined on bit strings of length  $n$ , is a natural multi-objective analogue of the single-objective JUMP problem, which might be the multimodal problem most studied in single-objective runtime analysis. The JUMP problem (and the two objectives of the ONEJUMPZEROJUMP problem) come with a difficulty parameter  $k \in [1..n] := \{1, \dots, n\}$ , which is the width of the valley of low fitness around

the global optimum. Consequently, typical hillclimbers at some point need to flip the right  $k$  bits, which is difficult already for moderate sizes of  $k$ . For the multi-objective ONEJUMPZEROJUMP problem the situation is similar. Here the Pareto front is not a connected set in the search space  $\{0, 1\}^n$ , but there are solutions which can only be reached from other points on the Pareto front by flipping  $k$  bits, which creates a challenge similar to the single-objective case.

*Our results:* We conduct a mathematical runtime analysis of the NSGA-II algorithm on the ONEJUMPZEROJUMP problem with jump sizes  $k \in [2.. \frac{1}{4}n]$ . We allow that  $k$  is functionally dependent on  $n$  and let all asymptotic notation be with respect to  $n$ . Since the runtimes we observe are at least exponential in  $k$ , the restriction of  $k \leq \frac{1}{4}n$ , done mostly to avoid some not very interesting technicalities, is not a harsh restriction. As *runtime*, we consider the number of fitness evaluations until the full Pareto front (that is, at least one individual for each Pareto-optimal objective value) is contained in the parent population of the NSGA-II. As in [13], we assume that the population size  $N$  of the NSGA-II is sufficiently large, here at least four times the size of the Pareto front (since a population size equal to the Pareto front size does not suffice to find the Pareto front even of the simple ONEMINMAX problem [13], this assumption appears justified). We regard the NSGA-II with four different ways to select the parents (each individual once (“fair selection”), uniform,  $N$  independent binary tournaments, and  $N$  binary tournaments from two random permutations of the population (“two-permutation tournament scheme”)), with bit-wise mutation with mutation rate  $\frac{1}{n}$ , and, for the theoretical analyses, without crossover. We prove that this algorithm on the ONEJUMPZEROJUMP problem with jump size  $k$  has an expected runtime of at most  $(1 + o(1))KNn^k$ , where  $K$  is a small constant depending on the selection method. Hence for  $N = \Theta(n)$ , the NSGA-II satisfies the same asymptotic runtime guarantee of  $O(n^k)$  as the (mostly relevant in theory) algorithm global SEMO (GSEMO), for which a runtime guarantee of  $(1 + o(1))1.5e(n - 2k + 3)n^k$  was shown in [8].

Since it has been observed (first time in [6], first time for MOEAs in [8]) that a heavy-tailed mutation operator called *fast mutation* can significantly speed up leaving local optima, we also regard the NSGA-II with this mutation operator. Similar to previous works, we manage to show a runtime guarantee which is lower by a factor of  $k^{\Omega(k)}$  (see Theorem 3 for a precise statement of this result). This result suggests that the NSGA-II, similar to many other algorithms, profits from fast mutation when local optima need to be left.

## 2 Previous Works

For reasons of space, we only briefly mention the most relevant previous works. For a more detailed account of the literature, we refer to these works or the (unfortunately not too recent) survey [3].

The first mathematical runtime analysis of the NSGA-II [13] showed that this algorithm can efficiently find the Pareto front of the ONEMINMAX and LOTZ bi-objective problems when the population size  $N$  is at least some constant factor

larger than the size of the Pareto front (which is  $n+1$  for these problems). In this case, once an objective value of the Pareto front is covered by the population, it remains so for the remaining run of the algorithm. This is different when the population size is only equal to the size of the Pareto front. Then such values can be lost, and this effect is strong enough that for an exponential number of iterations a constant fraction of the Pareto front is not covered [13]. Nevertheless, also in this case the NSGA-II computes good approximations of the Pareto front as the first experiments in [13] and a deeper analysis in [12] show.

The most recent work [2] extends [13] in several directions. (i) For the NSGA-II using crossover, runtime guarantees for the ONEMINMAX, COCZ, and LOTZ problems are shown which agree with those in [13]. (ii) By assuming that individuals with identical objective value appear in the same or inverse order in the sortings used to compute the crowding distance, the minimum required population size is lowered to  $2(n+1)$ . (iii) A stochastic tournament selection is proposed that reduces the runtimes by a factor of  $\Theta(n)$  on LOTZ and  $\Theta(\log n)$  on the other two benchmarks.

The ONEMINMAX, COCZ, and LOTZ benchmarks are all composed of two unimodal objectives, namely functions isomorphic to the benchmarks ONEMAX and LEADINGONES from single-objective EA theory. The theory of MOEA has strongly focused on such benchmarks, a benchmark composed of multimodal objectives was only proposed and analyzed in [8].

Besides the definition of the ONEJUMPZEROJUMP problem, the main results in that work are that the SEMO algorithm cannot optimize this benchmark, that the GSEMO takes time  $O((n-2k+3)n^k)$  (where the implicit constants can be chosen independent of  $n$  and  $k$ ), and that the GSEMO with fast mutation with power-law exponent  $\beta > 1$  succeeds in time  $O((n-2k+3)k^{-k+\beta-0.5}n^k(\frac{n}{n-k})^{n-k})$  (where the implicit constant can be chosen depending on  $\beta$  only). A slightly weaker, but still much better bound than for bit-wise mutation was shown for the GSEMO with the stagnation detection mechanism of Rajabi and Witt [11] (we omit the details for reasons of space).

### 3 Preliminaries

#### 3.1 The NSGA-II Algorithm

We only give a brief overview of the algorithm here due to space constraints, and refer to [5] for a more detailed description of the general algorithm and to [13] for more details on the particular version of the NSGA-II we regard.

The algorithm starts with a random initialization of a parent population of size  $N$ . At each iteration,  $N$  children are generated from the parent population via a mutation method, and  $N$  individuals among the combined parent and children population survive to the next generation based on their ranks in the non-dominated sorting and, as tie-breaker, the crowding distance.

Ranks are determined recursively. All individuals that are not strictly dominated by any other individual have rank 1. Given that individuals of rank  $1, \dots, i$

are defined, individuals of rank  $i + 1$  are those only strictly dominated by individuals of rank  $i$  or smaller. Clearly, individuals of lower ranks are preferred.

The crowding distance, denoted by  $\text{cDis}(x)$  for an individual  $x$ , is used to compare individuals of the same rank. To compute the crowding distances of individuals of rank  $i$  with respect to a given objective function  $f_j$ , we first sort the individuals in ascending order according to their  $f_j$  objective values. The first and last individuals in the sorted list have infinite crowding distance. For the other individuals, their crowding distance is the difference between the objective values of its left and right neighbors in the sorted list, normalized by the difference of the minimum and maximum values. The final crowding distance of an individual is the sum of its crowding distances with respect to each objective function.

At each iteration, the critical rank  $i^*$  is the rank such that if we take all individuals of ranks smaller than  $i^*$ , the total number of individuals will be less than or equal to  $N$ , but if we also take all individuals of rank  $i^*$ , the total number of individuals will be over  $N$ . Thus, all individuals of rank smaller than  $i^*$  survive to the next generation, and for individuals of rank  $i^*$ , we take the individuals with the highest crowding distance, breaking ties randomly, so that in total exactly  $N$  individuals are kept.

### 3.2 The OneJumpZeroJump Benchmark

Let  $n \in \mathbb{N}$  and  $k = \lfloor 2..n/4 \rfloor$ . The bi-objective function  $\text{ONEJUMPZEROJUMP}_{n,k} = (f_1, f_2) : \{0, 1\}^n \rightarrow \mathbb{R}^2$  is defined by

$$f_1(x) = \begin{cases} k + |x|_1, & \text{if } |x|_1 \leq n - k \text{ or } x = 1^n, \\ n - |x|_1, & \text{else;} \end{cases}$$

$$f_2(x) = \begin{cases} k + |x|_0, & \text{if } |x|_0 \leq n - k \text{ or } x = 0^n, \\ n - |x|_0, & \text{else.} \end{cases}$$

The aim is to maximize both  $f_1$  and  $f_2$ . The first objective is the classical  $\text{JUMP}_{n,k}$  function. It has a valley of low fitness around its optimum, which can be crossed only by flipping the  $k$  correct bits, if no solutions of lower fitness are accepted. The second objective is isomorphic to the first, with the roles of zeroes and ones exchanged. According to Theorem 2 of [8], the Pareto set of the  $\text{ONEJUMPZEROJUMP}_{n,k}$  function is  $S^* = \{x \in \{0, 1\}^n \mid |x|_1 = [k..n-k] \cup \{0, n\}\}$ , and the Pareto front  $F^*$  is  $\{(a, 2k+n-a) \mid a \in [2k..n] \cup \{k, n+k\}\}$ , which means the size of the front is  $n-2k+3$ . We define the inner part of the Pareto set by  $S_I^* = \{x \mid |x|_1 \in [k..n-k]\}$ , the outer part of the Pareto set by  $S_O^* = \{x \mid |x|_1 \in \{0, n\}\}$ , the inner part of the Pareto front by  $F_I^* = f(S_I^*) = \{(a, 2k+n-a) \mid a \in [2k..n]\}$ , and the outer part of the Pareto front by  $F_O^* = f(S_O^*) = \{(a, 2k+n-a) \mid a \in \{k, n+k\}\}$ .

## 4 Runtime Analysis for the NSGA-II

In this section, we prove our runtime guarantees for the NSGA-II, first with bit-wise mutation with mutation rate  $\frac{1}{n}$  (Subsection 4.1), then with fast mutation (Subsection 4.2).

The obvious difference to the analysis for ONEMINMAX in [13] is that with ONEJUMPZEROJUMP, individuals with between one and  $k - 1$  zeroes or ones are not optimal. Moreover, all these individuals have a very low fitness in both objectives. Consequently, such individuals usually will not survive into the next generation, which means that the NSGA-II at some point will have to generate the all-ones string from a solution with at least  $k$  zeroes (unless we are extremely lucky in the initialization of the population). This difference is the reason for the larger runtimes and the advantage of the fast mutation operator.

A second, smaller difference which however cannot be ignored in the mathematical proofs is that the very early populations of a run of the algorithm may contain zero individual on the Pareto front. This problem had to be solved also in the analysis of LOTZ in [13], but the solution developed there required that the population size is at least 5 times the size of the Pareto front (when tournament selection was used). For ONEJUMPZEROJUMP, we found a different argument to cope with this situation that, as all the rest of the proof, only requires a population size of at least 4 times the Pareto front size.

We start with a few general observations that apply to both cases. A crucial observation, analogous to a similar statement in [13], is that with sufficient population size, objective values of rank-1 individuals always survive to the next generation.

**Lemma 1.** *Consider one iteration of the NSGA-II algorithm optimizing the ONEJUMPZEROJUMP<sub>n,k</sub> benchmark, with population size  $N \geq 4(n - 2k + 3)$ . If in some iteration  $t$  the combined parent and offspring population  $R_t$  contains an individual  $x$  of rank 1, then the next parent population  $P_{t+1}$  contains an individual  $y$  such that  $f(y) = f(x)$ . Moreover, if an objective value on the Pareto front appears in  $R_t$ , it will be kept in all future iterations.*

*Proof.* Let  $F_1$  be the set of rank-1 individuals in  $R_t$ . To prove the first claim, we need to show that for each  $x \in F_1$ , there is a  $y \in P_{t+1}$  such that  $f(x) = f(y)$ . Let  $S_{1.1}, \dots, S_{1.|\mathcal{F}_1|}$  be the list of individuals in  $F_1$  sorted by ascending  $f_1$  values and  $S_{2.1}, \dots, S_{2.|\mathcal{F}_1|}$  be the list of individuals sorted by ascending  $f_2$  values, which were used to compute the crowding distances. Then there exist  $a \leq b$  and  $a' \leq b'$  such that  $[a..b] = \{i \mid f_1(S_{1.i}) = f_1(x)\}$  and  $[a'..b'] = \{i \mid f_2(S_{2.i}) = f_2(x)\}$ . If any one of  $a = 1$ ,  $a' = 1$ ,  $b = |\mathcal{F}_1|$ , or  $b' = |\mathcal{F}_1|$  is true, then there is an individual  $y \in F_1$  satisfying  $f(y) = f(x)$  of infinite crowding distance. Since there are at most  $4 < N$  individuals of infinite crowding distance,  $y$  is kept in  $P_{t+1}$ . So consider the case that  $a, a' > 1$  and  $b, b' < |\mathcal{F}_1|$ . By the definition of the crowding distance, we have that  $\text{cDis}(S_{1.a}) \geq \frac{f_1(S_{1.a+1}) - f_1(S_{1.a-1})}{f_1(S_{1.|\mathcal{F}_1|}) - f_1(S_{1.1})} \geq \frac{f_1(S_{1.a}) - f_1(S_{1.a-1})}{f_1(S_{1.|\mathcal{F}_1|}) - f_1(S_{1.1})}$ . Since  $f_1(S_{1.a}) - f_1(S_{1.a-1}) > 0$  by the definition of  $a$ , we have  $\text{cDis}(S_{1.a}) > 0$ . Similarly, we have  $\text{cDis}(S_{1.a'}), \text{cDis}(S_{1.b}), \text{cDis}(S_{1.b'}) > 0$ . For  $i \in [a + 1..b - 1]$

and  $S_{1,i} = S_{2,j}$  for some  $j \in [a' + 1..b' - 1]$ , we have that  $f_1(S_{1,i-1}) = f_1(x) = f_1(S_{1,i+1})$  and  $f_2(S_{2,j-1}) = f_2(x) = f_2(S_{2,j+1})$ . So  $\text{cDis}(S_{1,i}) = 0$ . Therefore, for each  $f(x)$  value, there are at most 4 individuals with the same objective value and positive crowding distances. By Corollary 6 in [8],  $|F_1| \leq n - 2k + 3$ . So the number of rank-1 individuals with positive crowding distances is at most  $4(n - 2k + 3) \leq N$  and therefore they will all be kept in  $P_{t+1}$ .

The second claim then follows since if  $x \in R_t$  and  $f(x)$  is on the Pareto front, we have  $x \in F_1$ . By the first claim,  $x \in P_{t+1}$  and therefore  $x \in R_{t+1}$ . The same reasoning applies for all future iterations.  $\square$

For our analysis, we divide a run of the NSGA-II algorithm optimizing the ONEJUMPZEROJUMP $_{n,k}$  benchmark into the following stages.

- *Stage 1:*  $P_t \cap S_I^* = \emptyset$ . In this stage, the algorithm tries to find the first individual with objective value in  $F_I^*$ .
- *Stage 2:* There exists a  $v \in F_I^*$  such that  $v \notin f(P_t)$ . In this stage, the algorithm tries to cover the entire set  $F_I^*$ .
- *Stage 3:*  $F_I^* \subseteq f(P_t)$ , but  $F_O^* \not\subseteq f(P_t)$ . In this stage, the algorithm tries to find the extremal values of the Pareto front.

By Lemma 1, once the algorithm has entered a later stage, it will not go back to an earlier stage. Thus, we can estimate the expected number of iterations needed by the NSGA-II algorithm by separately analyzing each stage.

A mutation method studied in [13] is to flip one bit selected uniformly at random. For reasons of completeness, we prove in the following lemma the natural result that the NSGA-II with this mutation operator with high probability is not able to cover the full Pareto front of the ONEJUMPZEROJUMP $_{n,k}$  benchmark.

**Lemma 2.** *With probability  $1 - N \exp(-\Omega(n))$ , the NSGA-II algorithm using one-bit flips as mutation operator does not find the full Pareto front of the ONEJUMPZEROJUMP $_{n,k}$  benchmark, regardless of the runtime.*

*Proof.* Since  $k \leq n/4$ , a simple Chernoff bound argument shows that a random initial individual is in  $S_I^*$  with probability  $1 - \exp(-\Omega(n))$ . By a union bound, we have  $P_0 \subseteq S_I^*$  with probability  $1 - N \exp(-\Omega(n))$ . We argue that in this case, the algorithm can never find an individual in  $S_O^*$ .

We observe that any individual in  $S_I^*$  strictly dominates any individual in the gap regions of the two objectives, that is, with between 1 and  $k - 1$  zeroes or ones. Consequently, in any population containing at least one individual from  $S_I^*$ , such a gap individual can never have rank 1, and the only rank 1 individuals are those on the Pareto front. Hence if  $P_t$  for some iteration  $t$  contains only individuals on the Pareto front,  $P_{t+1}$  will do so as well.

By induction and our assumption  $P_0 \subseteq S_I^*$ , we see that the parent population will never contain an individual with exactly one one-bit. Since only from such a parent the all-zeroes string can be generated (via one-bit mutation), we will never have the all-zeroes string in the population.  $\square$

In the light of Lemma 2, the one-bit flip mutation operator is not suitable for the optimization of ONEJUMPZEROJUMP. We therefore do not consider this operator in the following runtime analyses.

#### 4.1 Runtime Analysis for the NSGA-II Using Bit-Wise Mutation

In this section, we analyze the complexity of the NSGA-II algorithm when mutating each bit of each selected parent with probability  $\frac{1}{n}$ . We consider four different ways of selecting the parents for mutation: (i) fair selection (selecting each parent once), (ii) uniform selection (selecting one parent uniformly at random for  $N$  times), (iii) via  $N$  independent tournaments (for  $N$  times, uniformly at random sample 2 different parents and conduct a binary tournament between the two, i.e., select the one with the lower rank and, in case of tie, select the one with the larger crowding distance, and, in case of tie, select one randomly), and (iv) via a two-permutation tournament scheme (generate two random permutations  $\pi_1$  and  $\pi_2$  of  $P_t$  and conduct a binary tournament between  $\pi_j(2i-1)$  and  $\pi_j(2i)$  for all  $i \in [1..N/2]$  and  $j \in \{1, 2\}$ ; this is the selection method used in Deb's implementation of the NSGA-II when ignoring crossover [5]).

**Lemma 3.** *Using population size  $N \geq 4(n - 2k + 3)$ , bit-wise mutation for variation, and any parent selection method, stage 1 needs in expectation at most  $e(\frac{4k}{3})^k$  iterations.*

*Proof.* Suppose  $x$  is selected for mutation during one iteration of stage 1 and  $|x|_1 = i$ . Then  $i < k$  or  $i > n - k$ . If  $i < k$ , then the probability of obtaining an individual with  $k$  1-bits is at least  $\binom{n-i}{k-i}(\frac{1}{n})^{k-i}(1 - \frac{1}{n})^{n-(k-i)} \geq (\frac{n-i}{n(k-i)})^{k-i}(1 - \frac{1}{n})^{n-1} > \frac{1}{e}(\frac{3}{4(k-i)})^{k-i} \geq \frac{1}{e}(\frac{3}{4k})^k$  (where the second to last inequality uses the assumption that  $i < k \leq \frac{n}{4}$ ). If  $i > n - k$ , then the probability of obtaining an individual with  $n - k$  1-bits is at least  $\binom{i}{i-(n-k)}(\frac{1}{n})^{i-(n-k)}(1 - \frac{1}{n})^{2n-i-k} \geq (\frac{i}{n(i-n+k)})^{i-n+k}(1 - \frac{1}{n})^{n-1} > \frac{1}{e}(\frac{3}{4(i-n+k)})^{i-n+k} \geq \frac{1}{e}(\frac{3}{4k})^k$  (where the second to last inequality uses the assumption that  $i > n - k \geq \frac{3n}{4}$ ). Hence each iteration with probability at least  $\frac{1}{e}(\frac{3}{4k})^k$  marks the end of stage 1. Consequently, stage 1 ends after in expectation at most  $(\frac{1}{e}(\frac{3}{4k})^k)^{-1} = e(\frac{4k}{3})^k$  iterations.  $\square$

For the remaining two stages, we first regard the technically easier fair and uniform selection methods.

**Lemma 4.** *Using population size  $N \geq 4(n - 2k + 3)$ , selecting parents using fair or uniform selection, and using bit-wise mutation for variation, stage 2 needs in expectation  $O(n \log n)$  iterations.*

For reasons of space, we omit the proof, which is very similar to the corresponding part of the analysis on ONEMINMAX [13].<sup>1</sup> Different arguments, naturally, are needed in the following analysis of stage 3.

**Lemma 5.** *Using population size  $N \geq 4(n - 2k + 3)$  and bit-wise mutation for variation, stage 3 needs in expectation at most  $2en^k$  iterations if selecting parents using fair selection, and  $2\frac{e^2}{e-1}n^k$  iterations if using uniform selection.*

<sup>1</sup> For reasons of space, the formal proof of this result could not be included in this extended abstract. The reviewers can find such omitted materials in the appendix should they wish to consult these additional details.



*Proof.* Consider one iteration  $t$  of stage 3. We know that there is an  $x \in P_t$  such that  $|x|_1 = k$ . Denote the probability that  $x$  is selected at least once to be mutated in this iteration by  $p_1$ . Conditioning on  $x$  being selected, denote the probability that all  $k$  1-bits of  $x$  are flipped in this iteration by  $p_2$ . Then the probability of generating  $0^n$  in this iteration is at least  $p_1 p_2$ . Since by Lemma 1,  $x$  is kept for all future generations, we need at most  $\frac{1}{p_1 p_2}$  iterations to obtain  $0^n$ . With fair selection, we have  $p_1 = 1$  and with uniform selection,  $p_1 = 1 - (1 - \frac{1}{N})^N \geq 1 - \frac{1}{e}$ . On the other hand,  $p_2 = (\frac{1}{n})^k (1 - \frac{1}{n})^{n-k} \geq \frac{1}{en^k}$ . So the expected number  $\frac{1}{p_1 p_2}$  of iterations to obtain  $0^n$  is bounded by  $en^k$  if using fair selection, and by  $\frac{e^2}{e-1} n^k$  if using uniform selection. The case for obtaining  $1^n$  is symmetrical. Therefore, the expected total number of iterations needed to cover the extremal values of the Pareto front is at most  $2en^k$  if using fair selection, and  $2\frac{e^2}{e-1} n^k$  if using uniform selection.  $\square$

Combining the lemmas, we immediately obtain the runtime guarantee.

**Theorem 1.** *Using population size  $N \geq 4(n - 2k + 3)$ , selecting parents using fair or uniform selection, and mutating using bit-wise mutation, the NSGA-II needs in expectation at most  $(1 + o(1))KNn^k$  fitness evaluations to cover the entire Pareto front of the ONEJUMPZEROJUMP $_{n,k}$  benchmark, where  $K = 2e$  for fair selection and  $K = 2\frac{e^2}{e-1}$  for uniform selection.*

We note that in the above result, we have given explicit values for the leading constant  $K$  to show that it is not excessively large, but we have not tried to optimize this constant. In fact, it is easy to see that the 2 could be replaced by 1.5 by taking into account that the expected time to find the first extremal point is only half the time to find a particular extremal point. Further small improvements of the leading constant would be possible as well. Since we have no non-trivial lower bounds at the moment, we find it too early to optimize the constants.

We now turn to the case where the mutating parents are chosen using one of two ways of *binary tournaments*, namely, via  $N$  independent tournaments and the two-permutation tournament scheme.

**Theorem 2.** *Using population size  $N \geq 4(n - 2k + 3)$ , selecting parents using  $N$  independent tournaments or the two-permutation tournament scheme, and mutating using bit-wise mutation, the NSGA-II takes in expectation at most  $(1 + o(1))KNn^k$  fitness evaluations to cover the entire Pareto front of the ONEJUMPZEROJUMP $_{n,k}$  benchmark, where  $K = 2\frac{e^2}{e-1}$  if using  $N$  independent tournaments, and  $K = \frac{8}{3}e$  if using the two-permutation tournament scheme.*

The proof of this result follows the outline of the proof of Theorem 1, but needs some technical arguments from [13] on the probability that an individual next to an uncovered spot on the Pareto front is chosen to be mutated.

#### 4.2 Runtime Analysis for the NSGA-II Using Fast Mutation

We now consider the NSGA-II with heavy-tailed mutation, i.e., the mutation operator proposed in [6] and denoted by  $\text{MUT}^\beta(\cdot)$  in [8], a work from which we shall heavily profit in the following.

Let  $\beta > 1$  be a constant (typically below 3). Let  $D_{n/2}^\beta$  be the distribution such that if a random variable  $X$  follows the distribution, then  $\Pr[X = \alpha] = (C_{n/2}^\beta)^{-1} \alpha^{-\beta}$  for all  $\alpha \in [1..n/2]$ , where  $n$  is the size of the problem and  $C_{n/2}^\beta := \sum_{i=1}^{n/2} i^{-\beta}$ . In an application of the mutation operator  $\text{MUT}^\beta(\cdot)$ , first an  $\alpha$  is chosen according to the distribution  $D_{n/2}^\beta$  (independent from all other random choices of the algorithm) and then each bit of the parent is flipped independently with probability  $\alpha/n$ . Let  $x \in \{0, 1\}^n$ ,  $y \sim \text{MUT}^\beta(x)$ , and  $H(x, y)$  denote the Hamming distance between  $x$  and  $y$ . Then, by Lemma 13 of [8], we have

$$P_j^\beta := \Pr[H(x, y) = j] = \begin{cases} (C_{n/2}^\beta)^{-1} \Theta(1) & \text{for } j = 1; \\ (C_{n/2}^\beta)^{-1} \Omega(j^{-\beta}) & \text{for } j \in [2..n/2]. \end{cases}$$

**Theorem 3.** *Using population size  $N \geq 4(n - 2k + 3)$ , selecting parents using fair or uniform selection, and mutating with the  $\text{MUT}^\beta(\cdot)$  operator, the NSGA-II takes at most  $(1 + o(1)) \frac{1}{P_k^\beta} N K \binom{n}{k}$  fitness evaluations in expectation to cover the entire Pareto front of the ONEJUMPZEROJUMP $_{n,k}$  benchmark, where  $K = 2$  for fair selection,  $K = \frac{2e}{e-1}$  for uniform selection and selection via  $N$  independent binary tournaments, and  $K = \frac{8}{3}$  for the two-permutation binary tournament scheme.*

Noting that  $\binom{n}{k}$  is by a factor of  $k^{\Omega(k)}$  smaller than  $n^k$ , whereas  $1/P_k^\beta$  is only  $O(k^\beta)$ , we see that the runtime guarantee for the heavy-tailed operator is by a factor of  $k^{\Omega(k)}$  stronger than our guarantee for bit-wise mutation. Without a lower bound on the runtime in the bit-wise setting, we cannot claim that the heavy-tailed algorithm is truly better, but we strongly believe so (we do not see a reason why the NSGA-II with bit-wise mutation should be much faster than what our upper bound guarantees).

We note that it is easy to prove a lower bound of  $\Omega(n^k)$  for the runtime of the NSGA-II with bit-wise mutation (this is a factor of  $N$  below our upper bound, which stems from pessimistically assuming that in each iteration,  $N$  times a parent is selected that has a  $\Theta(n^k)$  chance of generating an extremal point of the Pareto front). For  $k$  larger than, say,  $\log(N)$ , this weak lower bound would suffice to show that the heavy-tailed NSGA-II is asymptotically faster. We spare the details and hope that at some time, we will be able to prove tight lower bounds for the NSGA-II.

We omit the formal proof of Theorem 3, which is not too different from the proofs of Theorems 1 and 2, for reasons of space.

When  $k \leq \sqrt{n}$ , the runtime estimates above can be estimated further as follows. In [8], it was shown that

$$P_i^\beta \geq \begin{cases} \frac{\beta-1}{e^\beta} & \text{for } i = 1; \\ \frac{\beta-1}{4\sqrt{2\pi}e^{8\sqrt{2}+13}\beta} i^{-\beta} & \text{for } i \in [2..\lfloor\sqrt{n}\rfloor]. \end{cases}$$

Also, for  $k \leq \sqrt{n}$ , a good estimate for the binomial coefficient is  $\binom{n}{k} \leq \frac{n^k}{k!}$  (losing at most a constant factor of  $e$ , and at most a  $(1 + o(1))$ -factor when  $k = o(\sqrt{n})$ ). Hence the runtime estimate from Theorem 3 for  $k \leq \sqrt{n}$  becomes

$$(1 + o(1))K \frac{4\sqrt{2\pi}e^{8\sqrt{2}+13}\beta}{\beta-1} N k^\beta \frac{n^k}{k!},$$

which is a tight estimate of the runtime guarantee of Theorem 3 apart from constants independent of  $n$  and  $k$ . In any case, this estimate shows that for moderate values of  $k$ , our runtime guarantee for the heavy-tailed NSGA-II is better by a factor of  $\Theta(k!k^{-\beta})$ , which is substantial already for small values of  $k$ .

## 5 Experiments

To complement our theoretical results, we also experimentally evaluate the runtime of the NSGA-II algorithm on the ONEJUMPZEROJUMP benchmark.

**Settings:** We implemented the algorithm as described in Section 3 in Python (we will make the code available upon acceptance). We use the following settings.

- Problem size  $n$ : 20 and 30.
- Jump size  $k$ : 3.
- Population size  $N$ : In our theoretical analysis, we have shown that with  $N = 4(n - 2k + 3)$ , the algorithm is able to recover the entire Pareto front. To further explore the effect of the population size, we conduct experiments with this population size, with half this size, and with twice this size, that is, for  $N \in \{2(n - 2k + 3), 4(n - 2k + 3), 8(n - 2k + 3)\}$ .
- Parent selection: For simplicity, we only experiment with using  $N$  independent binary tournaments.
- Mutation operator: Following our theoretical analysis, we consider two mutation operators, namely bit-wise mutation (flipping each bit with probability  $\frac{1}{n}$ ) and fast mutation, that is, the heavy-tailed mutation operator  $\text{MUT}^\beta(\cdot)$ .
- Number of independent repetitions per setting: 20. This number is a compromise between the longer runtimes observed on a benchmark like ONEJUMPZEROJUMP and the not very concentrated runtimes (for most of our experiments, we observed a corrected sample standard deviation between 50% and 80% of the mean, which fits to our intuition that the runtimes are dominated by the time to find the two extremal points of the Pareto front). We will conduct more runs for the revised version of this work.

**Table 1.** Average runtime of the NSGA-II with bit-wise mutation and heavy-tailed mutation operator on the ONEJUMPZEROJUMP benchmark with  $k = 3$ .

	$n = 20$		$n = 30$	
	Bit-wise	Heavy-tailed	Bit-wise	Heavy-tailed
$N = 2(n - 2k + 3)$	255432	196542	1590794	622507
$N = 4(n - 2k + 3)$	358445	173077	2159444	1199853
$N = 8(n - 2k + 3)$	457130	280255	2413919	2359594

**Experimental results:** Table 1 contains the average runtime (number of fitness evaluations done until the full Pareto front is covered) of the NSGA-II algorithm when using bit-wise mutation and the heavy-tailed mutation operator. The most obvious finding is that the heavy-tailed mutation operator already for these small problem and jump sizes gives significant speed-ups.

While our theoretical results are valid only for  $N \geq 4(n - 2k + 3)$ , our experimental data suggests that also with the smaller population size  $N = 2(n - 2k + 3)$  the algorithm is able to cover the entire Pareto front of the ONEJUMPZEROJUMP benchmark. We suspect that this is because even though theoretically there could be 4 individuals in each generation with the same objective value and positive crowding distances, empirically this happens relatively rarely and the expected number of individuals with the same objective value and positive crowding distances is closer to 2. We also note that with a larger population, e.g.,  $N = 8(n - 2k + 3)$ , naturally, the runtime increases, but usually by significantly less than a factor of two. This shows that the algorithm is able to profit somewhat from the larger population size.

**Crossover:** Besides fast mutation, recent research has detected two further mechanisms that can speed up the runtime of evolutionary algorithms on (single-objective) jump functions, namely the stagnation-detection mechanism of Rajabi and Witt [11] and crossover [4]. We are relatively optimistic that stagnation detection, as together with the global SEMO algorithm [8], can provably lead to runtime improvements, but we recall from [8] that the implementation of stagnation detection is less obvious for MOEAs. For that reason, we ignore this approach here and immediately turn to crossover, given that no results proving a performance gain from crossover for the NSGA-II exist, and there are clear suggestions on how to use it in [4].

Inspired by [4], we propose and experimentally analyze the following variant of the NSGA-II. The basic algorithm is as above. In particular, we also select  $N$  parents via independent tournaments. We partition these into pairs. For each pair, with probability 90%, we generate two intermediate offspring via a 2-offspring uniform crossover (that is, for each position independently, with probability 0.5, the first child inherits the bit from the first parent, and otherwise from the second parent; the bits from the two parents that are not inherited by the first child make up the second child). We then perform bit-wise mutation on these two intermediate offspring. With the remaining 10% probability, mutation is performed directly on the two parents.

**Table 2.** Average runtime of the NSGA-II with bit-wise mutation and crossover on the ONEJUMPZEROJUMP benchmark with  $k = 3$ 

	$n = 20$	$n = 30$	$n = 40$
$N = 2(n - 2k + 3)$	67208	237449	844310
$N = 4(n - 2k + 3)$	50810	186019	509053
$N = 8(n - 2k + 3)$	68687	272722	501084

Table 2 contains the average runtimes for this algorithm. We observe that crossover leads to massive speed-ups (which allows us to also conduct experiments for problem size  $n = 40$ ). More detailedly, comparing the runtimes for  $n = 30$  and bit-wise mutation (which is fair since the crossover version also uses this mutation operator), the crossover-based algorithm only uses between 8% and 15% percent of the runtime of the mutation-only algorithm.

We note that different from the case without crossover, with  $N = 2(n - 2k + 3)$ , the algorithm consistently takes more time than with  $N = 4(n - 2k + 3)$ . We suspect that the smaller population size makes it less likely that the population contains two parents from which crossover can create a profitable offspring.

## 6 Conclusions and Future Works

In this first mathematical runtime analysis of the NSGA-II on a bi-objective problem composed of multimodal objectives, we have shown that the NSGA-II with a sufficient population size performs well on the ONEJUMPZEROJUMP benchmark and profits from heavy-tailed mutation, all comparable to what was shown before for the GSEMO algorithm.

Due to the more complicated population dynamics of the NSGA-II, we could not prove an interesting lower bound. For this, it would be necessary to understand how many individuals with a particular objective value are in the population – note that this number is trivially one for the GSEMO, which explains why for this algorithm lower bounds could be proven [8]. Understanding better the population dynamics of the NSGA-II and with this possibly proving good lower bounds is clearly an interesting, though possibly challenging direction for future research.

A second interesting direction is to analyze how the NSGA-II with crossover optimizes the ONEJUMPZEROJUMP benchmark. Our (preliminary) experiments show clearly that crossover can lead to significant speed-ups here. Again, we currently do not have the methods to analyze this algorithm (note that the only previous work [2] regarding the NSGA-II with crossover does not obtain faster runtimes from crossover – thus it sufficed to exclude that crossover is harmful). To show that crossover leads to a speed-up, as in the analysis of speed-ups from crossover in the single-objective setting [4], again a good understanding of the population dynamics is necessary.

## References

1. Auger, A., Doerr, B. (eds.): Theory of Randomized Search Heuristics. World Scientific Publishing (2011)
2. Bian, C., Qian, C.: Running time analysis of the non-dominated sorting genetic algorithm II (NSGA-II) using binary or stochastic tournament selection. CoRR **abs/2203.11550** (2022)
3. Brockhoff, D.: Theoretical aspects of evolutionary multiobjective optimization. In: Auger, A., Doerr, B. (eds.) Theory of Randomized Search Heuristics, pp. 101–140. World Scientific Publishing (2011)
4. Dang, D., Friedrich, T., Kötzing, T., Krejca, M.S., Lehre, P.K., Oliveto, P.S., Sudholt, D., Sutton, A.M.: Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation* **22**, 484–497 (2018)
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* **6**, 182–197 (2002)
6. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Genetic and Evolutionary Computation Conference, GECCO 2017. pp. 777–784. ACM (2017)
7. Doerr, B., Neumann, F. (eds.): Theory of Evolutionary Computation—Recent Developments in Discrete Optimization. Springer (2020), also available at <https://cs.adelaide.edu.au/~frank/papers/TheoryBook2019-selfarchived.pdf>
8. Doerr, B., Zheng, W.: Theoretical analyses of multi-objective evolutionary algorithms on multi-modal objectives. In: Conference on Artificial Intelligence, AAI 2021. pp. 12293–12301. AAI Press (2021)
9. Jansen, T.: Analyzing Evolutionary Algorithms – The Computer Science Perspective. Springer (2013)
10. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity. Springer (2010)
11. Rajabi, A., Witt, C.: Self-adjusting evolutionary algorithms for multimodal optimization. In: Genetic and Evolutionary Computation Conference, GECCO 2020. pp. 1314–1322. ACM (2020)
12. Zheng, W., Doerr, B.: Better approximation guarantees for the NSGA-II by using the current crowding distance. In: Genetic and Evolutionary Computation Conference, GECCO 2022. ACM (2022), also available at <https://arxiv.org/abs/2203.02693>
13. Zheng, W., Liu, Y., Doerr, B.: A first mathematical runtime analysis of the Non-Dominated Sorting Genetic Algorithm II (NSGA-II). In: Conference on Artificial Intelligence, AAI 2022. AAI Press (2022), preprint at <https://arxiv.org/abs/2112.08581>
14. Zhou, A., Qu, B.Y., Li, H., Zhao, S.Z., Suganthan, P.N., Zhang, Q.: Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation* **1**, 32–49 (2011)

## A Appendix

This appendix contains the mathematical proofs which had to be omitted in the main paper for reasons of space. They are given here only in case a reviewer wants to access them. The paper without this appendix is our submission to PPSN and we feel that it is perfectly accessible without this appendix. Nevertheless, upon acceptance, we will make a version including this appendix available on the arxiv preprint server.

*Proof (of Lemma 4).* Let  $x \in P_t$  be an individual such that  $|x|_1 = v \in [k..n - k]$ . Denote the probability that  $x$  is selected as the parent at least once by  $p$ , the probability that the result of mutating  $x$  gives us a  $y$  such that  $|y|_1 = v + 1$  by  $p_v^+$  and the probability that the result of mutating  $x$  gives us a  $y$  such that  $|y|_1 = v - 1$  by  $p_v^-$ . Then, since by Lemma 1,  $f(x)$  is kept in  $f(P_t)$  for all iterations after it first appears, the expected number of iterations needed to obtain a  $y$  such that  $|y|_1 = v + 1$  is at most  $\frac{1}{pp_v^+}$ . Similarly, the expected number of iterations needed to obtain a  $y$  such that  $|y|_1 = v - 1$  is at most  $\frac{1}{pp_v^-}$ .

Consider one iteration  $t$  of stage 2. We know that there is  $x \in P_t$  such that  $|x|_1 = v \in [k..n - k]$ . Then the expected number of iterations needed to obtain objective values  $(k + v + 1, n + k - v - 1), (k + v + 2, n + k - v - 2), \dots, (n, 2k)$  is at most  $\sum_{i=v}^{n-k} \frac{1}{pp_i^+}$ . Similarly, the expected number of iterations needed to obtain objective values  $(k + v - 1, n + k - v + 1), (k + v - 2, n + k - v + 2), \dots, (2k, n)$  is at most  $\sum_{i=k}^v \frac{1}{pp_i^-}$ . As a result, the number of iterations needed to cover  $F_t^*$  is at most  $\sum_{i=v}^{n-k} \frac{1}{pp_i^+} + \sum_{i=k}^v \frac{1}{pp_i^-}$ . With fair selection, we have  $p = 1$ , and with uniform selection,  $p = 1 - (1 - \frac{1}{N})^N \geq 1 - \frac{1}{e}$ . Flipping each bit with probability  $\frac{1}{n}$ , we have  $p_v^+ = \frac{n-v}{n}(1 - \frac{1}{n})^{n-1} \geq \frac{n-v}{en}$  and  $p_v^- = \frac{v}{n}(1 - \frac{1}{n})^{n-1} \geq \frac{v}{en}$ . So the expected number of iterations is at most  $\frac{ne^2}{e-1}(\sum_{i=k}^{n-v} \frac{1}{i} + \sum_{i=k}^v \frac{1}{i}) = O(n \log n)$ .  $\square$

*Proof (of Theorem 2).* Since we aim for an asymptotic statement, we shall always assume that  $n$ , and consequently  $N$ , are sufficiently large.

From Lemma 3 we know that stage 1 takes at most  $e(\frac{4k}{3})^k$  iterations in expectation.

The analysis for stage 2 is similar to that in Lemma 4, but due to the different parent selection method, we need to find a new lower bound for  $p$ , the probability of selecting a suitable parent. Consider an iteration  $t$  of stage 2, let  $V = f(P_t)$ , and let  $v_{\min} = \min\{f_1(x) \mid x \in P_t\}$  and  $v_{\max} = \max\{f_1(x) \mid x \in P_t\}$ . Also, define  $V^+ = \{(v_1, v_2) \in V \mid \exists y \in P_t : (f_1(y), f_2(y)) = (v_1 + 1, v_2 - 1)\}$  and  $V^- = \{(v_1, v_2) \in V \mid \exists y \in P_t : (f_1(y), f_2(y)) = (v_1 - 1, v_2 + 1)\}$ . Then by Lemma 3 of [13], for any  $(v_1, v_2) \in V \setminus (V^+ \cap V^-)$ , there is at least one individual  $x \in P_t$  with  $f(x) = (v_1, v_2)$  and  $\text{cDis}(x) \geq \frac{2}{v_{\max} - v_{\min}}$ . We call such individuals “desirable individuals”. By Lemma 4 of [13], for any  $(v_1, v_2) \in V^+ \cap V^-$ , there are at most two individuals  $x \in P_t$  with  $f(x) = (v_1, v_2)$  and  $\text{cDis}(x) \geq \frac{2}{v_{\max} - v_{\min}}$ . Since  $|F_t^* \setminus \{(2k, n), (n, 2k)\}| = n - 2k - 1$ , there are at most  $2(n - 2k - 1)$  such individuals. By the proof of Lemma 1, there are at most 16 individuals with

objectives values  $(k, n+k)$ ,  $(2k, n)$ ,  $(n, 2k)$ , or  $(n+k, k)$  and positive crowding distance. Therefore, there are at least  $(N-1) - 2(n-2k-1) - 16 \geq \frac{N}{2} - 9$  individuals that, if chosen as the opponent of a desirable individual, the winner of the binary tournament is a desirable individual. Thus, with  $N$  independent tournaments, the probability that a desirable individual participates and wins in a binary tournament is at least  $p = 1 - (1 - \frac{1}{N} \frac{\frac{N}{2} - 9}{N-1})^N = 1 - (1 - \frac{1}{2N} \frac{N-18}{N-1})^N$ , which is an increasing function for  $N > 0$ . So for  $N \geq 4(n-2k+3) \geq 28$ , we have  $p \geq 0.17$ . With the two-permutation tournament scheme, the probability is at least  $\frac{\frac{N}{2}-9}{N-1} = \frac{N-18}{2(N-1)}$ . Following the proof of Lemma 4, the expected number of iterations when using  $N$  independent binary tournaments is at most  $\frac{en}{0.17} (\sum_{e-1}^{n-v} \frac{1}{i} + \sum_{i=k}^v \frac{1}{i}) = O(n \log n)$ , and when using the two-permutation tournament scheme is at most  $\frac{2en(N-1)}{N-18} (\sum_{i=k}^{n-v} \frac{1}{i} + \sum_{i=k}^v \frac{1}{i}) = O(n \log n)$ , where  $v$  is the number of 1-bits in the first  $x \in S_I^*$  found by the algorithm.

The analysis for stage 3 is similar to that in Lemma 5. Let  $x$  be an individual in  $P_t$  such that  $|x|_1 = k$ . Suppose  $0^n \notin P_t$ , then  $x$  has the largest  $f_2$  value in rank-1 individuals of  $P_t$ . Therefore  $\text{cDis}(x) = \infty$  and if  $x$  is selected to participate in a binary tournament, in the worst case it wins with probability  $\frac{1}{2}$ . If using  $N$  independent binary tournaments, the probability of  $x$  chosen as a mutating parent is  $1 - (1 - \frac{1}{2} \frac{2}{N})^N \geq 1 - \frac{1}{N}$ . With the two-permutation tournament scheme, the probability is  $1 - (\frac{1}{2})^2 = \frac{3}{4}$ . Following the proof of Lemma 5, the number of iterations to obtain  $0^n$  in stage 3 is at most  $\frac{1}{(1-\frac{1}{e}) \frac{1}{en^k}} = \frac{e^2}{e-1} n^k$  using  $N$  independent tournaments, and  $\frac{1}{\frac{3}{4} \frac{1}{en^k}} = \frac{4}{3} en^k$  under the two-permutation tournament scheme. The case for covering  $1^n$  is symmetrical. So in total stage 3 needs in expectation at most  $2 \frac{e^2}{e-1} n^k$  iterations using  $N$  independent tournaments, and  $\frac{8}{3} en^k$  iterations under the two-permutation tournament scheme.  $\square$

*Proof (of Theorem 3).* The analysis for stage 1 is similar to the proof of Lemma 3. Now the probability of obtaining an individual with  $k$  1-bits from an individual with  $i < k$  1-bits is  $p_1 = \frac{\binom{n-i}{k-i} P_{k-i}^\beta}{\binom{n}{k-i}} = \frac{(n-i)!(n-k+i)!}{(n-k)!n!} P_{k-i}^\beta > (\frac{n-k}{n})^i P_{k-i}^\beta > (\frac{3}{4})^k P_{k-i}^\beta$ . Similarly, the probability of obtaining an individual with  $n-k$  1-bits from an individual with  $i > n-k$  1-bits is  $p_2 = \frac{\binom{i-(n-k)}{i-(n-k)} P_{i-(n-k)}^\beta}{\binom{i}{i-(n-k)}} = \frac{i!(2n-i-k)!}{(n-k)!n!} P_{i-(n-k)}^\beta > (\frac{n-k}{n})^{n-i} P_{k-i}^\beta > (\frac{3}{4})^k P_{k-i}^\beta$ . Since  $\beta > 1$ ,  $C_{n/2}^\beta \leq \frac{\beta}{\beta-1}$ . Consequently  $p_1, p_2 > (\frac{3}{4})^k \frac{\beta-1}{\beta} \Omega(k^{-\beta})$ . So the number of iterations needed in expectation is at most  $O((\frac{4}{3})^k k^\beta)$ .

The analysis of stage 2 follows the one in Lemma 4. Now, from an individual with  $i$  1-bits,  $p_i^+$ , the probability of generating an individual with one more 1 is  $\frac{n-i}{n} P_1^\beta$ , and  $p_i^-$ , the probability of generating an individual with one more 0 is  $\frac{i}{n} P_1^\beta$ . So the number of iterations needed in expectation for stage 2 is at most

$$\sum_{i=v}^{n-k} \frac{1}{pp_i^+} + \sum_{i=k}^v \frac{1}{pp_i^-} \leq \frac{2en}{(e-1)P_1^\beta} (1 + \ln n) = O(n \log n).$$



The analysis of stage 3 is again similar to the one in Lemma 5 except that now we profit from the much larger probability to flip  $k$  bits. The probability of selecting an individual with  $k$  1-bits as a mutating parent,  $p_1$ , remains unchanged, and thus  $p_1 = 1$  with fair selection and  $p_1 \geq 1 - \frac{1}{e}$  with uniform selection. Now, the probability of generating an individual with 0 1-bits from an individual with  $k$  1-bits,  $p_2 = \frac{P_k^\beta}{\binom{n}{k}}$ . So the number of iterations needed in expectation to find  $0^n$  is at most  $\frac{\binom{n}{k}}{P_k^\beta}$  with fair selection, and  $\frac{e\binom{n}{k}}{(e-1)P_k^\beta}$  with uniform selection. Finding  $1^n$  is the same. So the total number of iterations needed in expectation is at most  $\frac{2\binom{n}{k}}{P_k^\beta}$  for fair selection and  $\frac{2e\binom{n}{k}}{(e-1)P_k^\beta}$  for uniform selection.

Combining the three stages, the number of fitness evaluations needed in total is at most  $(1 + o(1))\frac{1}{P_k^\beta}K\binom{n}{k}$  in expectation, where  $K = 2$  for fair selection,  $K = \frac{2e}{e-1}$  for uniform selection.

We omit the analysis for using binary tournaments to select the mutating parents, since it is very similar to what has been done so far.  $\square$