**Statement of Authorship**

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Passau, July 6, 2023

**Abstract**

A short summary of what is going on here.

**Deutsche Zusammenfassung**

Kurze Inhaltsangabe auf deutsch.

# Contents

# 1. Introduction

This chapter should contain

1. A short description of the thesis topic and its background.
2. An overview of related work in this field.
3. Contributions of the thesis.
4. Outline of the thesis.

# 2. Preliminaries

## 2.1. Notations

1. **RLS:** Randomised Local Search
2. **RSH:** Randomised Search Heuristic referring to all analysed Evolutionary algorithms
3. $n$**:** The input length of the problem
4. $w_i$**:** The $i$-th object of the input. If not mentioned otherwise the weights are sorted in non-increasing order so: $w_1 \geq w_2 \geq \ldots \geq w_{n-1} \geq w_n$
5. $W$**:** The sum of all objects: $\mathrm{W} = \sum_{i=1}^{n} w_i$
6. **bin:** When solving Partition a set of numbers is divided into two distinct subsets and in this paper both subsets are referred to as bins
7. $b_F$**:** The fuller bin (the bin with more total weight)
8. $b_E$**:** The emptier bin (the bin with less total weight)
9. $b_{w_i}$**:** The bin containing the object $w_i$
10. *opt***:** The optimal solution for a given partition instance.
11. $x$**:** A vector $x \in \{0,1\}^n$ describing a solution

# 3. Improving bounds on the RLS and (1+1) EA

## 3.1. Improving bounds on the RLS and the (1+1) EA

**Lemma 3.1.** *If $w_1 \geq \frac{W}{2}$ then the RLS and the (1+1) EA reach the optimal value in expected time $\Theta(n \log n)$*

*Proof.* The optimal solution is putting $w_1$ in one bin and all other elements in the other bin. So the problem is almost identical to OneMax/ZeroMax. A single bit flip of the first bit can only happen, if the emptier bin has a weight of at most $\frac{W-w_1}{2}$. After this flip the weight of the emptier bin is at least $\frac{W-w_1}{2}$ and therefore another single bit flip of $w_1$ can only happen before a different bit is flipped. After a different bit has been flipped, the RLS wont flip the first bit again, because it will never result in an improvement. So the run of the RLS can be divided into three phases:

Phase 1: The RLS behaves exactly like OneMax/ZeroMax and flips every bit to the opposite of the first bit (except for the first bit).

Phase 2: The RLS flips only the first bit or bits that do not result in an improvement.

Phase 3: The RLS behaves exactly like ZeroMax/OneMax and flips every bit to the opposite of the first bit (except for the first bit).

The expected length of the first phase is $\mathcal{O}(n)$ because the probability of flipping the first bit is at least $\frac{1}{n} \cdot (1 - \frac{1}{n})^{n-1} \geq \frac{1}{en}$ and therefore the expected time for such a step is at most $\mathcal{O}(\frac{1}{en}^{-1}) = \mathcal{O}(ne) = \mathcal{O}(n)$.
The length of the second phase is $\mathcal{O}(n)$ because the solution is either optimal or there is at least one bit that needs to be flipped for an optimal solution. Since the expected length of Phase 1 is $\mathcal{O}(n)$ the solution produced by the RLS won't be optimal in expectation due to the bound of $\Theta(n \log n)$ for OneMax/ZeroMax. This again results in expected time $\mathcal{O}(n)$.
The length of the third phase is identical to a run of the RLS on OneMax/ZeroMax where flips of the first bit are ignored as if it was already correctly flipped and therefore the expected time is $\Theta(n \log n)$

So the total expected time is $\mathcal{O}(n) + \mathcal{O}(n) + \Theta(n \log n) = \Theta(n \log n)$

The (1+1) EA can do multiple bit flips in a single step so the first bit can be flipped multiple times if the combined moved weight $y \leq b_F - b_E$. ***TODO: insert proof for (1+1) EA***. □

**Lemma 3.2.** *If $b_F \leq \frac{2}{3} \cdot W$ the approximation ratio is at most $\frac{4}{3}$*

*Proof.* $\frac{b_F}{opt} \leq \frac{(2/3)\cdot W}{opt} \leq \frac{(2/3)\cdot W}{(1/2)\cdot W} = \frac{4}{3}$, since $opt \geq \frac{W}{2}$ □

**Corollary 3.3.** *If $w_1 \geq \frac{W}{3}$ and $w_1$ is in the emptier bin, then the approximation ratio is at most $\frac{4}{3}$*

*Proof.* $w_1$ is in the emptier bin, so $b_F \leq W - w_1 \leq W - \frac{W}{3} = \frac{2W}{3}$ and with Lemma 3.2 the assumption follows. □

**Lemma 3.4.** *Any object of weight at most $v$ can be moved from $b_F$ to $b_E$ if $b_F - b_E \geq v$*

*Proof.* $b_F - b_E \geq v \Leftrightarrow b_F \geq b_E + v$, so after moving an object with weight at most $v$ from $b_F$ to $b_E$, the new weight of $b_E$ is at most the weight of $b_F$ before moving the object, thus the RSH accepts the step. □

**Corollary 3.5.** *The RLS is stuck in a local optima if $b_F - b_E < w_n$ holds and $b_F > opt$.*

*Proof.* A single bit flip of weight $v$ can only happen if $b_F - b_E \geq v$. If $b_F - b_E < w_n$ there is no weight which satisfies the condition and therefore no single bit flip is possible. Since the RLS can only move one bit at a time and only if it results in an improvement, the RLS is stuck. □

**Corollary 3.6.** *Every object $\leq \frac{W}{3}$ can be moved from $b_F$ to $b_E$ if $b_F \geq \frac{2W}{3}$*

*Proof.* $b_F \geq \frac{2W}{3} \Rightarrow b_E \leq W - \frac{2W}{3} \leq \frac{W}{3} \Rightarrow b_F - b_E \geq \frac{2W}{3} - \frac{W}{3} = \frac{W}{3}$ and with Lemma 3.4 the assumption follows. □

**Lemma 3.7.** *In expected Time $\mathcal{O}(n \log n)$ the weight of the fuller bin can be decreased to $\leq \frac{2W}{3}$ if every object besides the biggest in the fuller bin is at most $\frac{W}{3}$ and $w_1 \leq \frac{W}{2}$.*

*Proof.* In expected time $\mathcal{O}(n \log n)$ the RSH can move every object $\leq \frac{W}{3}$ to the emptier bin as long as $b_F \geq \frac{2W}{3}$ due to Corollary 3.6 and results for OneMax. So in expected Time $\mathcal{O}(n \log n)$ the solution can be shifted to $w_1$ being in one bin and all other objects in the other bin. The RSH will only stop moving the elements if the condition $b_F \geq \frac{2W}{3}$ is no longer satisfied (Corollary 3.6). If $w_1 \geq \frac{W}{3}$ and every object was moved to the bin without $w_1$, then $b_F = \max\{W - w_1, w_1\} = W - w_1 \leq \frac{2W}{3}$, because $w_1 \leq \frac{W}{2}$. So either the RSH moves all objects to the emptier bin or stops moving objects because $b_F < \frac{2W}{3}$ both resulting in $b_F \leq \frac{2W}{3}$. If $w_1$ is not in the fuller bin, then the result follows by Corollary 3.3. Now assume $w_1 < \frac{W}{3}$. In this case the RLS will move one object per step to the emptier bin. Each object has weight $< \frac{W}{3}$ and therefore one step can not decrease the weight of the fuller bin from $> \frac{2W}{3}$ to $\leq \frac{W}{3}$. If all objects except the biggest where moved the other bin, the other bin would have a weight of at least $W - w_1 > \frac{2W}{3}$. Therefore the RLS will find a solution with $b_F < \frac{2W}{3}$ before moving all elements from the first to the second bin. ***TODO: insert proof for (1+1) EA*** □

**Lemma 3.8.** *The RLS and the (1+1) EA reach an approximation ratio of at most $\frac{4}{3}$ in expected time $\mathcal{O}(n \log n)$ if $w_1 < W/2$*

*Proof.* If $w_1 + w_2 > \frac{2W}{3}$ after time $\mathcal{O}(n)$ $w_1$ and $w_2$ are separated and will remain separated afterwards (Proof by C.Witt [Die05]). From then on the following holds. If $w_1$ is in the emptier bin, then the result follows directly by Corollary 3.3. Otherwise all elements in the fuller bin except $w_1$ have a weight of at most $\frac{1}{3}$ and therefore the result follows by Lemma 3.7 and Lemma 3.2. If $w_1 + w_2 \leq \frac{2W}{3}$ the result follows directly by Lemma 3.7 and Lemma 3.2. □

**Corollary 3.9.** *The RLS and the (1+1) EA reach an approximation ratio of at most $\frac{4}{3}$ in expected time $\mathcal{O}(n \log n)$*

*Proof.* This follows directly from Lemma 3.1 and Lemma 3.8 □

## 3.2. Binomial distributed input

**Lemma 3.10.** *A binomial distributed input ~$B(m, p)$ has an optimal solution with high probability if n is large enough.*

*Proof.* Sketch:

- The initial distribution is likely rather close to the optimum
- The difference between the bins is probably not more than 10 expected values
- the large values

Consider a random separation of all values into two sets with equal size if $n$ is even or one set with one value more than the other if $n$ is odd. The sum X of one set is a sum of $\frac{n}{2} \cdot m$ independent Bernoulli trials with probability $p$. With Chernoff Bounds the following inequality follows:

$$\mathbb{P}(X \geq (\frac{n}{2} + \sqrt{\frac{n}{2}}) \cdot m \cdot p) = \mathbb{P}(X \geq (1 + \sqrt{\frac{2}{n}}) \cdot nmp) \leq e^{-mnp \cdot \sqrt{\frac{2}{n}}^2/3} = e^{-\frac{mp}{3}}$$

For $mp \geq 3$ the probability is less than $\frac{1}{e}$. Otherwise the input is rather trivial, since the numbers will be sharply concentrated around 3. There will also be many 1s because 1 is close to the expected value, making the possibility for an optimal solution even grater.
After moving $\mathcal{O}(\sqrt{\frac{2}{n}}/2)$ objects to the emptier set, the difference between the two sets is at most half the expected value $mp$ of a single value. From then on □

**Lemma 3.11.** *With high probability the RLS does not find an optimal solution for an input with distribution ~$B(m, p)$ if n and m are large enough.*

*Proof.* Sketch:

- There exists an optimal solution with high probability due to last lemma
- probability for a value to be very low is almost 0 if m is huge
- The RLS only moves one element per step and will step below bF-bE < wn without bF = opt being true
- -> RLS cant make another step and is stuck in a local optimum.

Due to Lemma 3.10 the input has an optimal solution with high probability. □

# 4. Heavy Tailed Mutations

## 4.1. Algorithms

Heavy tailed mutations are mutations that flip more than one bit in expectation. For the (1+1) EA this can be achieved by simply changing the mutation rate 1/n to c/n for any constant c.

For the RLS it is not that simple, as the RLS chooses a random bit and flips it. Instead of flipping c bits in every step there should be the possibility to flip different amounts of bits in every step. The standard RLS chooses a random neighbour with hamming distance one. So the heavy tailed version could simply choose neighbours that have a hamming distance larger than one. The selection should still be uniform random to keep the idea of the RLS intact. One possible way is to choose a random neighbour with hamming distance $\leq k$. The amount of neighbours with hamming distance x is given by $\binom{n}{x}$. For k = 4, this results in n neighbours with hamming distance 1, n(n-1)/2 neighbours with hamming distance 2, n(n-1)(n-2)/6 and n(n-1)(n-2)(n-3)/24. The probability to choose a random neighbour with hamming distance $x \leq k$ is given by

$$P(\text{RLS-N(k) flips x bits}) = \frac{\binom{n}{x}}{\sum_{i=1}^{k}\binom{n}{i}} = \frac{\mathcal{O}(n^x)}{\sum_{i=1}^{k}\mathcal{O}(n^i)} = \frac{\mathcal{O}(n^x)}{\mathcal{O}(n^k)} = \mathcal{O}(n^{x-k}) = \mathcal{O}(\frac{1}{n^{k-x}})$$

This variant of the RLS is likely to choose a neighbour with hamming distance k as the number of neighbours with hamming distance k rises with k for $k \leq n/2$. The probability of flipping only one bit is therefore $\mathcal{O}(\frac{1}{n^{k-1}})$. For some inputs flipping only one bit might be more optimal which is rather unlikely for this variant of the RLS.

**Algorithm 4.1:** (1+1) EA WITH STATIC MUTATION RATE

```
1 choose x uniform from {0,1}^n
2 while x not optimal do
3 |   x' ← x
4 |   flip every bit of x' with probability c/n
5 |   if f(x') ≤ f(x) then
6 |   |   x ← x'
```

**Algorithm 4.2:** RLS-N

```
choose x uniform from {0,1}^n
while x not optimal do
    x' ← uniform random neighbour of x with hamming distance ≤ k
    if f(x') ≤ f(x) then
        x ← x'
```

**Algorithm 4.3:** RLS-R

```
choose x uniform from {0,1}^n
while x not optimal do
    y ← uniform random value ∈ {1,...,k}
    x' ← uniform random neighbour of x with hamming distance y
    if f(x') ≤ f(x) then
        x ← x'
```

An alternative way of changing the RLS is to first choose $x \in 1, \ldots, k$ uniform random and then choose a neighbour with hamming distance $x$ uniform random. Here the probability of flipping $x \leq k$ is given by $1/k$, so the algorithm is much more likely to choose to flip only one bit.

Both variants of the RLS change at most k bits in each step and therefore only a constant amount of bits. For the (1+1) EA the algorithm will also flip mostly $\mathcal{O}(c)$ bits which is also constant. So neither of the new variants is likely to change up to n bits. Quinzan *et al.* therefore introduced another mutation operator in [FGQW18] called $pmut_\beta$. This operator chooses $k$ from a powerlaw distribution with parameter $\beta$ and then $k$ uniform random bits are flipped. This algorithm will mostly flip a small number of bits but occasionally up to n bits.

# 5. Experimental Results

In the following chapter the different variants of the RLS and the (1+1) EA are now analysed empirically for the best algorithm depending on the input. Additionally for most lemmas from the previous chapters there are also tests if they actually hold in practice.

## 5.1. Code

The complete java code used for all empirical studies is available on GitHub:
https://github.com/Err404NameNotFound/PartitionSolvingWithEAs.

### 5.1.1. The Algorithms

All different variants of the RLS function more or the less the same. They start with an initial random value and then optimise this one value int the loop. The loop can be summarised like this:

1. generate a number k of bits to be flipped (algorithm specific)
2. flip k random bits
3. evaluate fitness of the mutated individual
4. replace old value with new value if new value is better
5. repeat if not optimal

The (1+1) EA variants behave a differently on the first impression as there every bit is flipped independently with probability $c/n$. This can be seen as n independent Bernoulli trials with probability $c/n$. The amount of bits that are flipped is therefore binomial distributed and the algorithm can be implemented exactly as the versions of the RLS. The same holds for the pMut operator which generates a number k from a powerlaw distribution and then flips k bits. This leads to only one implementation of a partition solving algorithm which is not only given the input array of numbers but also a generator for the amount of bits to be flipped in each step. The random values for the amount of bits to be flipped are generated according to this table:

**Algorithm 5.1:** GenericPartitionSolver

```
choose x uniform random from {0,1}^n
while x not optimal do
    x' ← x
    k ← kGenerator.generate()
    flip k uniform random bits of x'
    if f(x') ≤ f(x) then
        x ← x'
```

| Algorithm | Returned value |
|---|---|
| RLS | 1 |
| RLS-N(k) | $y \in \{1, \ldots, k\}$ with probability $\frac{\binom{n}{y}}{\sum_{i=1}^{k}\binom{n}{i}}$ |
| RLS-R(k) | uniform random value $x \in \{1, \ldots, k\}$ |
| (1+1) EA | binomial distributed value from $\sim B(n, c/n)$ |
| pMut | random value generated from powerlaw distribution with parameter $\beta$ |

### 5.1.2. Random number generation

Java only provides a random number generator for uniform distributed values for any integer interval or random double values $\in [0, 1]$. For this project this does not suffice as for an efficient way of implementing the (1+1) EA or simply for generating a binomial distributed input another random number generator is needed. One of the needed distributions is a binomial distribution. The simplest way to generate a number $\sim B(m, p)$ would be to run a loop $m$ times and add 1 to the generated number if a uniform random value $\in [0, 1]$ is at most $p$. This works perfectly fine and generates numbers according to the distribution. With low values for p this approach is rather inefficient and especially for values of $p = 1/m$. The expected value in this case is 1 but generating a random number takes time $\mathcal{O}(n)$. Another more efficient way was implemented by StackOverflow user pjs on stackoverflow inspired by Devroyes method introduced in [Dev06]. This method has an expected running time of $mp$ which is equal to the expected value of the distribution. For the case of $p = 1/m$ this runs in expected constant time in comparison to $\mathcal{O}(n)$ for the naive way. This number generation was also used for the implementation of the (1+1) EA instead of running a for loop in every step.

**Algorithm 5.2:** Binomial random number generator

```
q ← ln(1.0 − p)
x ← 0
sum ← 0
while true do
    sum ← sum + ln(random())/(n − x)
    // random() generates a random value ∈ [0,1)
    if sum < q then
        return x
    x ← x + 1
```

The next generator needed is for exponentially distributed values. This generator is only necessary for the generation of exponentially distributed inputs but not for the algorithms itself. The easiest way to generate exponentially distributed values is the naive way:

generating a uniform random value until the generated value is greater than the probability. The expected running time of this algorithm is equal to the expected value of the distribution $1/p$. So this method is comparably effective to the approach used for binomial random number generation.

**Algorithm 5.3:** Exponential random number generator

```
1 sum ← 0
  // random() generates a random value ∈ [0, 1)
2 while random() < q do
3 |  sum ← sum + 1
4 return sum
```

The last generator need is for powerlaw distributed values. This generator is in contrast to the exponential number generator only needed for the algorithm with the $pmut_\beta$ mutation operator. The generator used for all experiments is also from stackoverflow. The user gnovice provided the following formula on this page on stackoverflow:

$$x = [(b^{n+1} - q^{n+1}) * y + a^{n+1}]^{1/(n+1)}$$

$a$ is the lower bound, $b$ the upper bound, $n$ the parameter of the distribution and $y$ the number generated uniform random $\in [0, 1)$. The idea behind the formula and the formula itself is explained in a mathworld page.

## 5.2. Do inputs have perfect partitions?

### 5.2.1. Binomial Inputs

Lemma 3.10 is only valid for larger n. In practice the bound is much smaller depending on the expected value of a single value. Another factor deciding how likely an input is to have a perfect partition is if n is even or odd. To determine the influence of both factors two experiments were conducted. The goal of the first experiment was to determine the influence of the array size to the input having a perfect partition and the fact if n is even or odd. So for every possible combination of $p \in \{0.1, 0.2, \ldots, 0.8, 0.9\}, m \in \{10, 100, 1000, 10^4, 10^5\}$ and $n \in \{2, 3, 4, \ldots, 19, 20\}$ 1000 randomly generated inputs of size $n$ were generated and tested for a perfect partition. Due to the small values for $n$ it was possible to brute force the results in a short amount of time. The results are visualised in figure 5.1 to figure 5.5. On the x-axis is the size of the input and on the y-axis the percentage of inputs that had a perfect partition. The different graphs in one figure resemble the different values of p used for generating the inputs. The graph for 0.1 resembles the percentage of inputs that had a perfect partition that were generated from the distribution $\sim B(m, 0.1)$ with $m$ being dependent on the figure. For figure 5.1 $m$ has the value 10.
It is easy to see that for small inputs sizes it is relevant if n is even or odd as all curves in figure 5.5 oscillate between 0% and 100% for $n \geq 14$. For uneven inputs the probability of a perfect partition decreases much more drastically with m as for even inputs because the expected value of a single number increases with m. If all values are much higher the small differences between the values can no longer even out the fact of one set has more elements than the other. The oscillation therefore increases with increasing m. For $n = 20$ all 1000 inputs had a perfect partition for every combination of $p$ and $m$ but for $n = 19$ only combinations where $mp \leq 300$ holds lead to at least one out of 1000 having a perfect partition. For expected values of up to $10^5$ it seems to be almost granted that an input of length 20 has a perfect partition if it is binomial distributed. Even for only

12 binomial generated values more than 50% of the inputs had a perfect partition (see figure 5.5). Another visible effect is the decreasing percentage with rising p. This may be a direct result of the value chosen for p but can also be an indirect result as the value for p changes the expected value for a constant m. The expected value may have an influence on the number of perfect partitions because it influences the highest value of the input. For uniform distributed inputs Borgs showed that the coefficient of #bits needed to encode the max value/$n$ has a huge impact on the number of perfect partitions [BCP01]. For a coefficient $< 1$ the probability of a perfect partition tends to 1 and for a coefficient $> 1$ it tends to 0. This was only proven for the uniform distributed input, but it might also hold for a binomial distributed input. This leads to the second experiment.

Figure 5.1.: Percentage of Binomial inputs with perfect partitions for m = 10

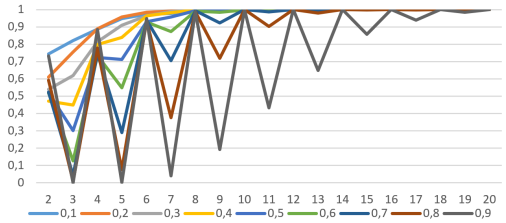



Figure 5.2.: Percentage of Binomial inputs with perfect partitions for m = 100

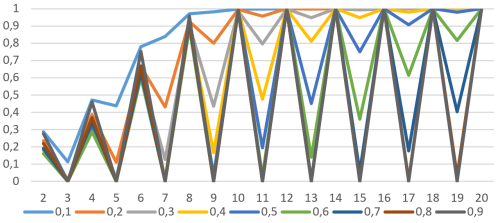



Figure 5.3.: Percentage of Binomial inputs with perfect partitions for m = 1000

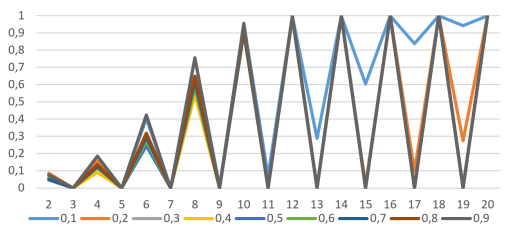



Figure 5.4.: Percentage of Binomial inputs with perfect partitions for m = 10000

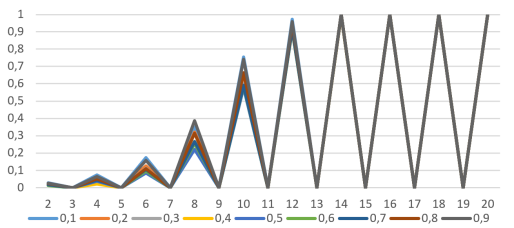
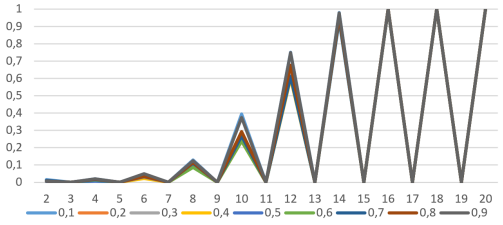



Figure 5.5.: Percentage of Binomial inputs with perfect partitions for m = 100000




In the second experiment the inputs were generated a bit differently. Here the goal was to keep the expected value fixed for any combination of $p$ and $n$ and set the value of $m$ to $e/p$ for all $e \in \{10, 20, 30, 40, 50, 100, 200, 500, 1000, 2000, 5000, 10000, 50000\}$ so that $E(X) = mp = e/p \cdot p = e$. With this setup the influence of the expected value is almost isolated from the other parameters. The probability is still linked to $p$ as $p$ also influences the variance $mp(1-p)$. By looking at figure 5.6 to figure 5.11 it seems as if the value of $p$ has a much smaller influence than the expected value. For a fixed expected value and a fixed input size a higher value for $p$ seems to only slightly increase the percentage of inputs with a perfect partition. The expected value influences the percentage significantly

more. For $p = 0.1, n = 14$ the value decreases from 100% at $E(X) = 10$ to below 20% at $E(X) = 50000$. For $p = 0.9$ the percentage only drops below 50% but still decreases by a factor of 2.

Figure 5.6.: Percentage of Binomial inputs with perfect partitions for p = 0.1

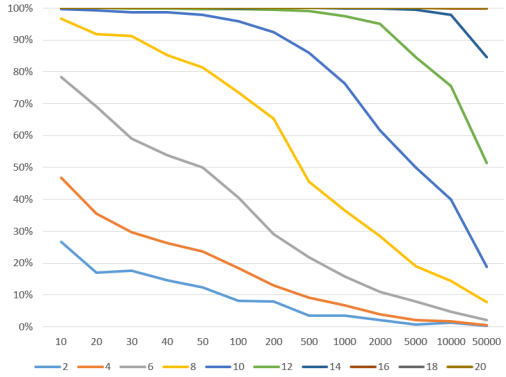



Figure 5.7.: Percentage of Binomial inputs with perfect partitions for p = 0.2

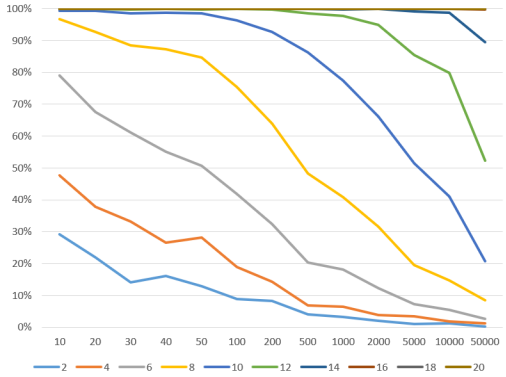



Figure 5.8.: Percentage of Binomial inputs with perfect partitions for p = 0.3

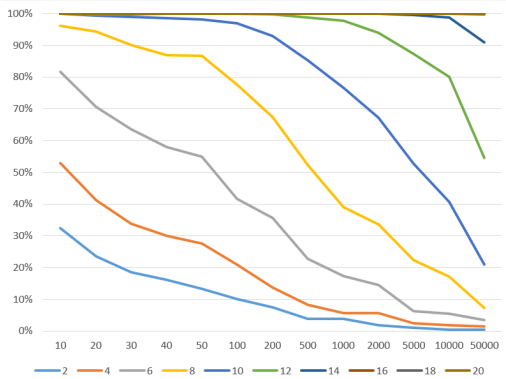



Figure 5.9.: Percentage of Binomial inputs with perfect partitions for p = 0.4

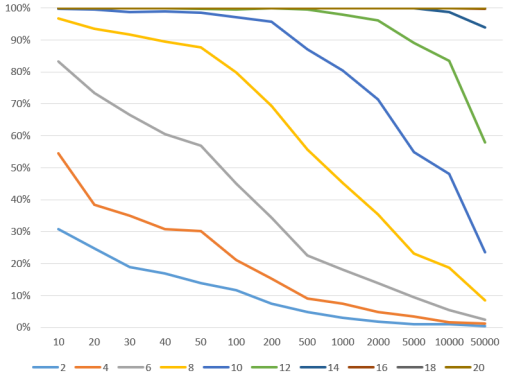


Figure 5.10.: Percentage of Binomial inputs with perfect partitions for p = 0.5

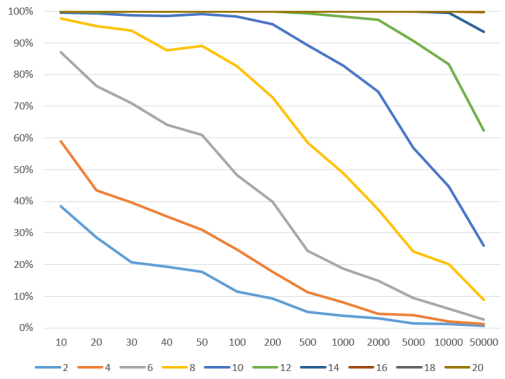
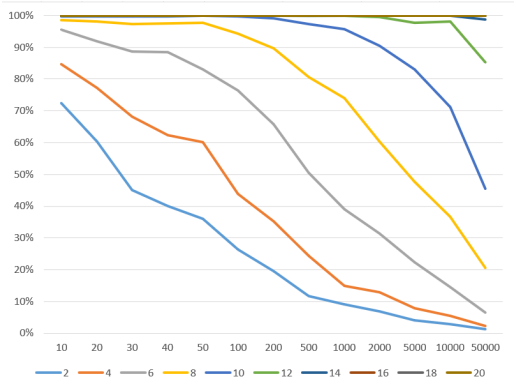



Figure 5.11.: Percentage of Binomial inputs with perfect partitions for p = 0.9




## 5.3. Binomial distributed inputs

### 5.3.1. RLS Comparison

### 5.3.2. (1+1) EA Comparison

### 5.3.3. pmut Comparison

### 5.3.4. Comparison of the best variants

## 5.4. Uniform distributed inputs

### 5.4.1. RLS Comparison

### 5.4.2. (1+1) EA Comparison

### 5.4.3. pmut Comparison

### 5.4.4. Comparison of the best variants

## 5.5. Exponential distributed inputs

### 5.5.1. RLS Comparison

### 5.5.2. (1+1) EA Comparison

### 5.5.3. pmut Comparison

### 5.5.4. Comparison of the best variants

## 5.6. OneMax Equivalent for PARTITION

This kind of input is more or less equivalent to the OneMax problem. All values except the last are either 1 or uniform random in any intervall. The last value is the sum of all other values. The optimal solution is therefore the 000...01 or the 111...01 string. So the best solution is almost identical to OneMax. In the previous chapter the O(nlogn) bound was proven for the (1+1) EA and the RLS. This seems to hold in practice: TODO: insert Graph showing RLS and (1+1) EA need time O(nlogn)

For OneMax the mutation rate of 1/n is proven to be optimal for the (1+1) EA (TODO insert cite). This seems to be also true for the equivalent for PARTITION. For Both the (1+1) EA and both new Variants of the RLS

# Bibliography


[BCP01] Christian Borgs, Jennifer Chayes, and Boris Pittel. Phase transition and finite-size scaling for the integer partitioning problem. *Random Structures & Algorithms*, 19(3-4):247–288, 2001.

[Dev06] Luc Devroye. Nonuniform random variate generation. *Handbooks in operations research and management science*, 13:83–121, 2006.

[Die05] Volker Diekert. *STACS 2005: 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2004, Proceedings*, volume 3404. Springer Science & Business Media, 2005.

[FGQW18] Tobias Friedrich, Andreas Göbel, Francesco Quinzan, and Markus Wagner. Evolutionary algorithms and submodular functions: Benefits of heavy-tailed mutations. *arXiv preprint arXiv:1805.10902*, 2018.

# Appendix

## A. Appendix Section 1

ein Bild

Figure A.1.: A figure