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 13, 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

```
choose x uniform from {0,1}^n
while x not optimal do
    x' ← x
    flip every bit of x' with probability c/n
    if f(x') ≤ f(x) then
        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 β 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 in 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 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**
  $x' \leftarrow x$
  $k \leftarrow$ kGenerator.generate()
  flip k uniform random bits of x'
  if $f(x') \leq f(x)$ **then**
    $x \leftarrow x'$

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 $y \in \{1, \ldots, k\}$
(1+1) EA	binomial distributed value from $\sim B(n, c/n)$
pMut	random value generated from powerlaw distribution with parameter β

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 less than 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 $\mathcal{O}(m)$. 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 $\mathcal{O}(mp)$ 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 $\mathcal{O}(m)$ 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

$q \leftarrow \ln(1.0 - p)$
$x \leftarrow 0$
$sum \leftarrow 0$
while *true* **do**
  $sum \leftarrow sum + \ln(\text{random}())/(n - x)$
  `// random() generates a random value` $\in [0, 1)$
  if $sum < q$ **then**
    return x
  $x \leftarrow x + 1$

The next generator needed is for geometric distributed values. This generator is only necessary for the generation of geometric distributed inputs but not for the algorithms itself. The easiest way to generate geometric distributed values is the naive way: generating a uniform random value until the generated value is at least as big as 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: GEOMETRIC 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 needed is for powerlaw distributed values. This generator is in contrast to the geometric number generator only needed for the algorithm with the $pmut_\beta$ mutation operator. This implementation 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 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 each 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 with values 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 for higher expected values 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

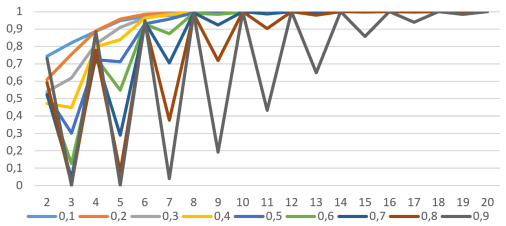


Figure 5.2.: Percentage of Binomial inputs with perfect partitions for m = 100

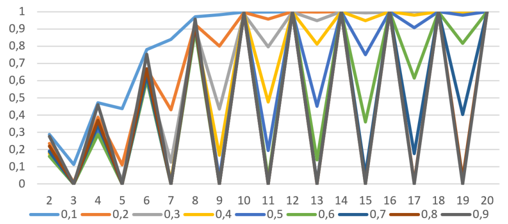


Figure 5.3.: Percentage of Binomial inputs with perfect partitions for m = 1000

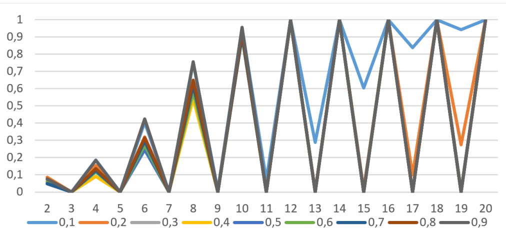


Figure 5.4.: Percentage of Binomial inputs with perfect partitions for m = 10000

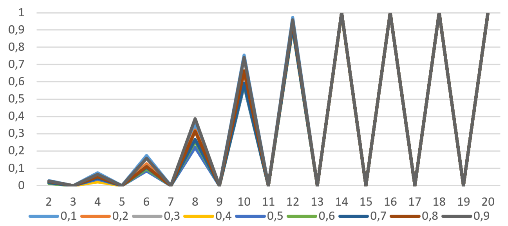
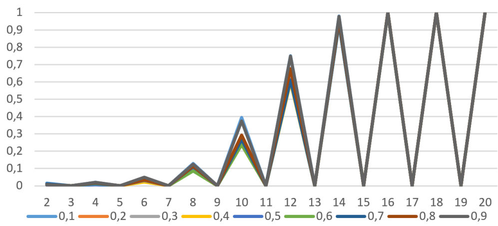


Figure 5.5.: Percentage of Binomial inputs with perfect partitions for m = 100000



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

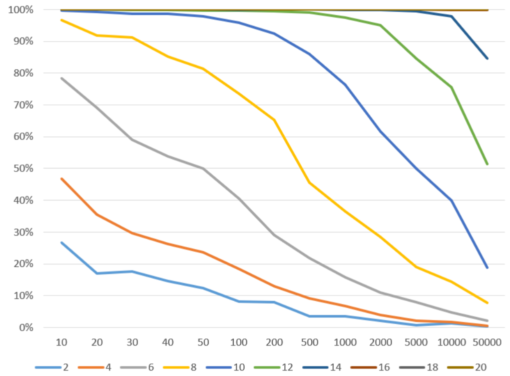


Figure 5.7.: Percentage of Binomial inputs with perfect partitions for p = 0.2

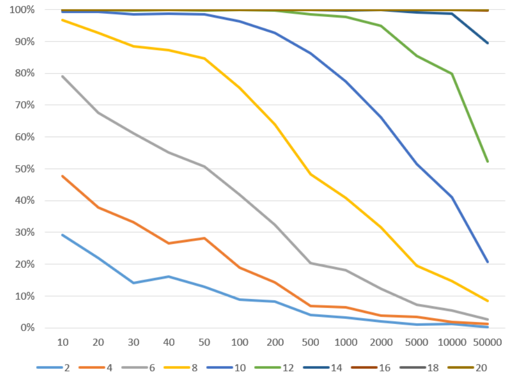


Figure 5.8.: Percentage of Binomial inputs with perfect partitions for p = 0.3

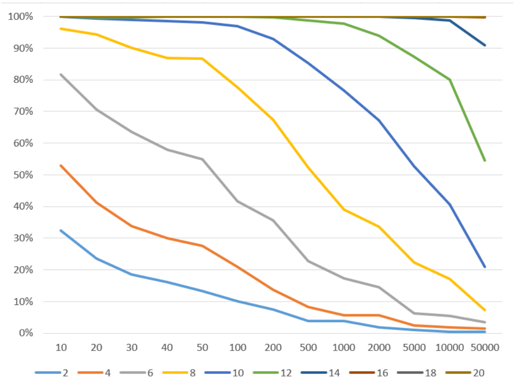
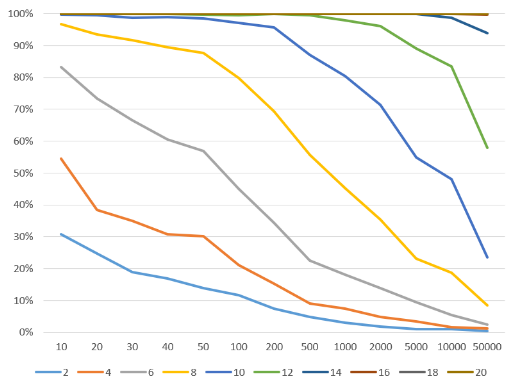


Figure 5.9.: Percentage of Binomial inputs with perfect partitions for p = 0.4



The last experiment showed that for $n = 20$ 1000/1000 inputs had a perfect partition. This raised the question of how the amount of perfect partition changes with changing values for m, p, n.

5.3. Binomial distributed values

In the following subsections the performance of the different algorithms is tested for different kinds of inputs. The exact distributions of the input are explained separately in each subsection. The procedure for each comparison is always the same. A random input is generated according to the distribution and then solved by every algorithm. All algorithms had the same two stopping conditions. The first was reaching a perfect partition and the second was taking more than $100 \cdot n \ln(n)$ steps. If either of these condition was met, the algorithm returned the current best solution. This step is repeated 1000 times. The results are presented in a table containing multiple statistics for each algorithm over all 1000 runs.

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

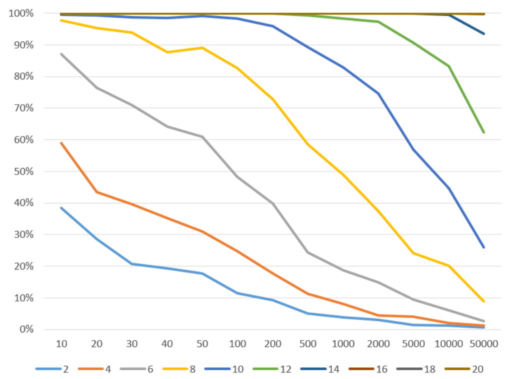
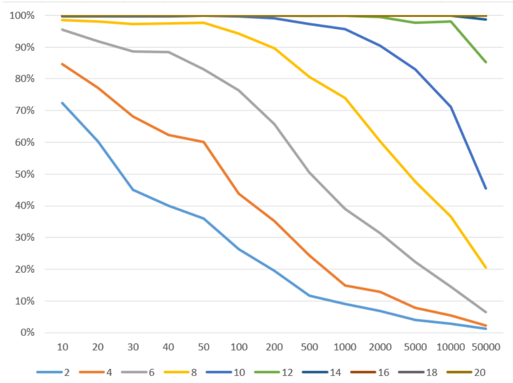


Figure 5.11.: Percentage of Binomial inputs with perfect partitions for p = 0.9

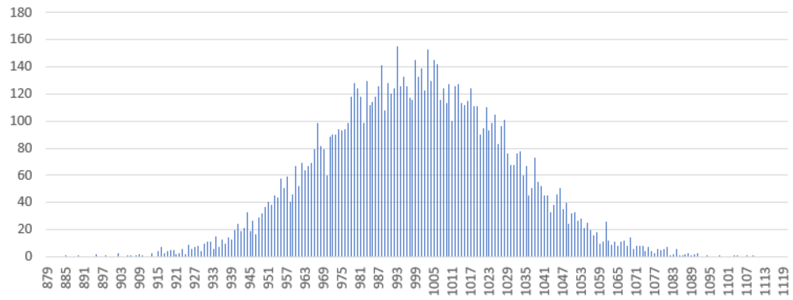


column name	meaning
algo type	type of algorithm (RLS, RLS-N, RLS-R, (1+1)EA or pmut)
algo param	parameter of the algorithm or '-' if it is the standard variant
avg mut/change	average #bits flipped for iterations leading to an improvement
avg mut/step	average #bits flipped for any iteration
total avg count	average #iterations for all runs
avg eval count	average #iterations of runs returning an optimal solution
max eval count	maximum #iterations of runs returning an optimal solution
min eval count	minimum #iterations of all runs
fails	number of runs that did not find an optimal solution
fail ratio	ratio of unsuccessful runs to all runs
avg fail dif	average value of $b_F - b_E$ for non-optimal solutions

Firstly the different variants of the RLS are compared with values of $k \in \{2, 3, 4\}$, then the performance of the (1+1) EA with static mutation rate c/n with $c \in 1, 2, 3, 5, 10, 50, 100$ and lastly the performance of the $pmut_\beta$ mutation operator with the parameter $\beta \in \{-1.25, -1.5, \dots, -2.75, -3.0, -3.25\}$. Additionally the best variants of each algorithm are compared in another 1000 runs.

The first analysed inputs are inputs following a binomial distribution $\sim B(m, p)$ as those inputs have been researched in the previous subsection. The results showed that the expected value of a single number is the main driver for the amount of perfect partitions the input has. The results also suggested the inputs tend to have more perfect partitions if the expected value is lower. The more perfect partitioned an input has relative to the number of all possible partitions, the more likely the different RSHs are to find one of those. Therefore researching inputs with higher expected values seems more interesting but generating higher values takes more time with a random number generator that needs $\mathcal{O}(mp)$ time. To keep the time for generating one set of numbers reasonable the values chosen for all tests are $m = 10000, p = 0.1, n = 10000$ with the expected value for a single number being $mp = 1000$. Figure 5.12 shows a random binomial distributed input of length $n = 10000$. All elements are sharply concentrated around the expected value with all values being at 1000 ± 200. So after the reaching a difference between the two bins of below $(1000 - 200)/2 = 400$ the algorithm can no longer achieve an improvement by flipping a single bit.

Figure 5.12.: Distribution of a random binomial input



5.3.1. RLS Comparison

algo type	RLS-N	RLS-N	RLS-R	RLS-R	RLS-R	RLS-N	RLS
algo param	n=2	n=4	r=2	r=4	r=3	n=3	-
avg mut/change	2.000	4.000	1.603	2.553	2.000	2.728	1.000
avg mut/step	2.000	4.000	1.500	2.500	1.999	3.000	1.000
total avg count	318	434	499	579	681	518,428	920,109
avg eval count	318	434	499	579	681	395,440	50
max eval count	1,648	3,243	3,094	3,737	4,717	917,134	50
min eval count	20	28	11	17	16	0	50
fails	0	0	0	0	0	234	999
fail ratio	0.000	0.000	0.000	0.000	0.000	0.234	0.999
avg fail dif	-	-	-	-	-	1	254

The RLS-N_2 seems to perform the best as it mostly switches two elements which works great for binomial distributed inputs. The same algorithm with $k = 4$ performs a bit worse but still good as switching 4 elements can be beneficial as well. The variant of RLS-N with $k = 3$ on the other hand does not reach the optimal solution in 23.4% of the inputs with an average difference of 1. It also needs 1000 times more iterations to find the optimal on average compared to the best algorithm RLS-N_2. The RLS-R variants behave mostly the same with $k = 2$ being the best, followed by $k = 4$ and $k = 3$. In this case the variant of $k = 3$ is by far not as bad as for the RLS-N because the probability of flipping 2 bits is 1/3 as compared to $\mathcal{O}(n^{-1})$ for the RLS-N. The RLS-R seem all to be good option for binomial inputs with values of $k \in \{2, 3, 4\}$. The RLS on the other hand performs by far the worst as it only moves one element at a time. It only managed to reach the optimal solution once for 1000 different inputs. The number of iterations for this input was only 50 so the RLS likely had a good initialisation with a few lucky steps leading directly to the optimum. For all other cases the average difference between the bins was 254 which is close to the average value of the values from 0 to $(1000 - 100)/2 = 450$. This is likely due to the RLS being unable to improve the solution once the current solution has a difference below half of the lowest value.

5.3.2. (1+1) EA Comparison

For the (1+1) EA the best static mutation rate seems to be $3/n$. The probability of flipping 2 or 4 bits as n goes to infinity for mutation rate $1/n$ approaches $13/24e \approx 0.199$, for $2/n$ approaches $8/3e^2 \approx 0.361$, for $3/n$ approaches $63/8e^3 \approx 0.392$, for $4/n$ approaches $56/3e^4 \approx 0.342$ and for $5/n$ approaches $77/2e^5 \approx 0.259$. So the highest probability has $c = 3$, followed by $c = 4$ and $c = 2$ then $c = 5$ and lastly $c = 1$. For higher values of c the probability decreases further as the expected number of flipped bits is c for mutation rate c/n.

algo type	EA-SM	EA-SM	EA-SM	EA-SM	EA	EA-SM	EA-SM	EA-SM
algo param	3/n	4/n	2/n	5/n	-	10/n	50/n	100/n
avg mut/change	3.101	3.968	2.343	4.859	1.698	9.732	49.544	99.494
avg mut/step	2.999	4.003	2.002	4.999	1.001	9.998	49.998	99.997
total avg count	646	701	706	857	1,123	1,508	8,175	15,485
avg eval count	646	701	706	857	1,123	1,508	8,175	15,485
max eval count	5,346	5,692	3,415	5,572	7,001	12,112	52,831	145,269
min eval count	23	4	30	9	23	14	27	69
fails	0	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-	-

The static mutation rate $3/n$ seems to perform the best with both $4/n$ and $2/n$ being a close second place. The next best values are $5/n$ and the standard $1/n$ both having a clear difference between each other and the better parameters. From then on the number of iterations rises monotonically with rising mutation rate. The higher mutation rates perform significantly worse but the still find a solution within the limit as opposed to the standard RLS. There is no clear winner but $\beta = -2.25$ had the best performance in this experiment it was used for the comparison of the best variants.

5.3.3. pmut Comparison

algo type	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut
algo param	-2.25	-2.00	-1.75	-2.50	-2.75	-3.00	-3.25	-1.50	-1.25
avg mut/change	3.822	6.266	14.371	2.804	2.347	1.995	1.843	38.318	92.365
avg mut/step	4.344	8.504	22.176	2.878	2.272	1.933	1.732	70.476	224.535
total avg count	652	668	675	688	697	718	758	785	1,050
avg eval count	652	668	675	688	697	718	758	785	1,050
max eval count	4,340	4,506	5,616	5,098	9,140	5,081	6,189	6,542	7,837
min eval count	14	4	9	12	27	10	11	21	7
fails	0	0	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-	-	-

For the $pmut_\beta$ mutation operator the choice of β seems to be much more insignificant than for the RLS or (1+1) EA. Here all values perform comparably good with only the value of $\beta = -1.25$ having a clear performance difference compared to next best value. All values of β reach an optimal solution in every case. The worst variant of the $pmut_\beta$ operator still performs much better than the worst value for the (1+1) EA and even better than the worst RLS variant.

5.3.4. Comparison of the best variants

algo type	RLS-N	EA-SM	pmut
algo param	n=2	3/n	-2.25
avg mut/change	2.000	3.092	3.965
avg mut/step	2.000	2.999	4.339
total avg count	302	677	691
avg eval count	302	677	691
max eval count	1,610	6,404	5,205
min eval count	9	33	17
fails	0	0	0
fail ratio	0.000	0.000	0.000
avg fail dif	-	-	-

For this setting of $m = 10000, p = 0.1, n = 10000$ the RLS-N with $k = 2$ performs better than the (1+1) EA and $pmut_\beta$ mutation for all values of c/n and β by a factor of at least 2. This is likely from the fact that this version of the RLS flips almost only two bits which seems to be close to optimal for this kind of input as there are many values close to the expected value which can be switched to make small adjustments to the fitness value. The (1+1) EA with $p = 3/n$ and $pmut_\beta$ algorithm with $\beta = -2.25$ perform almost the same. The (1+1) EA has a slightly lower average value but also has a higher minimum value and a higher maximum value. To further investigate which input perform best on all binomial distributed inputs now a comparison with different input lengths follows. The parameter of the distribution were not changed. The following table is the amount of runs in which the algorithms did not find an optimal solution within 50000 steps. The time limit was set 50000 because the algorithm normally reach the optimal solution within a few thousand steps. If the solutions is not found after 50000 steps, the algorithm is most likely stuck in a local optima which could only be left by flipping more bits than possible for the algorithm.

input size	20	50	100	500	1000	5000	10000
RLS-N(2)	231	22	3	0	0	0	0
RLS-N(4)	0	1	2	0	0	0	0
RLS-R (2)	243	4	0	0	0	0	0
RLS-R (4)	1	0	0	0	0	0	0
(1+1) EA (3/n)	0	0	0	0	0	0	0
pmut(-2.5)	0	0	0	0	0	0	0

The (1+1) EA and the $pmut_\beta$ algorithm always reach an optimal solution but the RLS does not. The variants that can only flip two steps per step perform significantly worse for small inputs. They are probably more likely to get stuck in a local optimum where a step flipping 4 bits or more would be necessary. So the RLS-N(2) does perform better for larger inputs but is much more likely to get stuck in a local optima. The next table contains the average number of iterations the algorithm needed to find an optimal solution for all runs where the algorithms managed to find an optimal solution. Here it still looks like the RLS-N(2) finds the solution with the lowest amount of steps, because the cases where the algorithm is stuck in a local optima are not contained in this table.

input size	20	50	100	500	1000	5000	10000
RLS-N(2)	164	257	337	256	257	293	310
RLS-N(4)	349	355	661	412	412	436	425
RLS-R (2)	267	435	408	428	442	452	496
RLS-R (4)	601	491	492	491	534	562	560
(1+1) EA (3/n)	716	570	569	580	614	625	650
pmut(-2.5)	1596	576	600	637	657	700	685

The next table contains the overall average amount of iterations for every run. So runs where no optimal was found add 50000 thousand to the sum of all iterations.

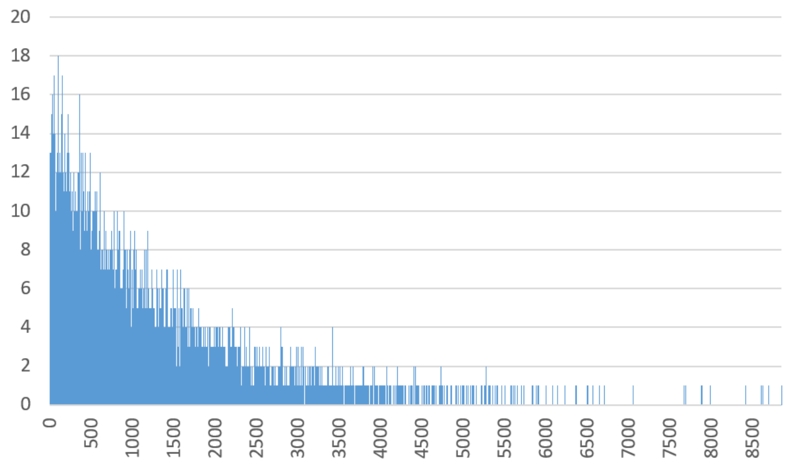
input size	20	50	100	500	1000	5000	10000
RLS-N(2)	11676	1351	486	256	257	293	310
RLS-N(4)	349	405	760	412	412	436	425
RLS-R (2)	12352	633	408	428	442	452	496
RLS-R (4)	650	491	492	491	534	562	560
(1+1) EA (3/n)	716	570	569	580	614	625	650
pmut(-2.5)	1596	576	600	637	657	700	685

Here the RLS-N(2) is only the best algorithm for values of $n \geq 500$. Below this bound choosing the (1+1) EA with static mutation rate $3/n$ is a saver choice as the (1+1) EA reaches an optimal solution for every input (in this experiment).

5.4. Geometric distributed values

For the geometric distribution the chosen default value is $p = 0.001$. This results in an expected value of 1000 which is the same as for the binomial distribution in the last subsection. This should make the results more comparable. Another parameter is the maximum value which was set to $10 \cdot E(X) = 10000$. Theoretically with very low probability the geometric distribution could result in an almost endless loop with bad luck. To prevent this issue the maximum value was inserted as an upper bound for the random number generator. Figure 5.13 shows that this maximum value does not change the input drastically as no value over 9000 was generated anyway. The span of all values is way higher than for the binomial distribution, although they have same expected value. Here the values are not in the interval $[800, 1200]$ but rather between 0 and the maximum value.

Figure 5.13.: Distribution of a random geometric input



The geometric distribution does not only have low values close or equal to 1 but also has the most values that are very small. This should lead to 1-bit flips being effective as the small values can remove the small differences. Because there are so many small values moving only one bit might be better than switching two elements.

5.4.1. RLS Comparison

algo type	RLS-R	RLS-R	RLS-R	RLS-N	RLS-N	RLS-N	RLS
algo param	r=2	r=3	r=4	n=2	n=3	n=4	-
avg mut/change	1.477	1.959	2.431	2.000	3.000	4.000	1.000
avg mut/step	1.500	2.001	2.501	2.000	3.000	4.000	1.000
total avg count	2,592	2,945	3,259	3,497	4,463	5,345	6,650
avg eval count	2,592	2,945	3,259	3,497	4,463	5,345	2,055
max eval count	19,845	23,932	28,532	23,824	30,881	41,600	25,889
min eval count	8	22	19	18	43	19	23
fails	0	0	0	0	0	0	5
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.005
avg fail dif	-	-	-	-	-	-	1

For these inputs the variants of the RLS perform differently to the binomial input. The only similarity is the RLS being the worst as the RLS is the only algorithm that did not find an optimal solution for every input. If the RLS did find an optimal solution in those 5 cases it would instead be the best RLS variant. The other algorithms are ranked by the probability of flipping only one bit. This means at first the three RLS-R variants from 2 to 3 to 4 and then the same for the RLS-N variants. So it does seem like moving mostly one element at once is better for the geometric input in comparison to two elements for the binomial distribution. In the 5 cases where the RLS did not find an optimal solution it was most likely stuck in a local optimum.

5.4.2. (1+1) EA Comparison

algo type	EA-SM	EA	EA-SM	EA-SM	EA-SM	EA-SM	EA-SM	EA-SM
algo param	2/n	-	3/n	4/n	5/n	10/n	50/n	100/n
avg mut/change	2.255	1.554	3.038	3.948	4.883	9.821	49.798	99.814
avg mut/step	2.000	1.000	3.000	4.001	5.000	9.999	49.998	100.001
total avg count	3,712	3,833	4,195	4,472	5,465	8,282	21,648	29,404
avg eval count	3,712	3,833	4,195	4,472	5,465	8,282	21,648	29,404
max eval count	39,593	53,450	33,598	42,449	55,717	65,522	149,048	281,857
min eval count	18	13	15	14	25	23	46	17
fails	0	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-	-

The results for the (1+1) EA are similar to the results of the RLS. From mutation rate $3/n$ on the performance decreases with rising mutation rate. The only part that does not fit into the theory of 1 bit flips being superior is the mutation rate $2/n$ performing better than the standard $1/n$. The average number of iterations for the standard (1+1) EA is only slightly higher than for the mutation rate $2/n$, so this might be just due to a too small runs of the algorithms. All variants reach an optimal solution within the given limit for the number of iterations.

5.4.3. pmut Comparison

algo type	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut
algo param	-3.25	-3.00	-2.50	-2.75	-2.25	-2.00	-1.75	-1.50	-1.25
avg mut/change	1.682	1.872	2.688	2.150	3.704	6.938	16.352	41.906	107.789
avg mut/step	1.730	1.936	2.918	2.267	4.355	8.463	22.369	70.989	225.029
total avg count	2,575	2,732	2,734	2,776	2,809	3,165	3,486	4,389	6,151
avg eval count	2,575	2,732	2,734	2,776	2,809	3,165	3,486	4,389	6,151
max eval count	73,911	34,215	75,791	42,620	25,352	31,966	37,725	50,454	55,022
min eval count	33	17	11	0	35	9	5	23	19
fails	0	0	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-	-	-

The results for the $pmut_\beta$ operator are even more clear than for the (1+1) EA. With increasing values (decreasing absolute values) for β the amounts of flipped bits per steps increases. The performance on the other hand decreases with rising values for β which fits into the theory of one bit flips being better for gemoetric distributed inputs. The only special case that does not support this theory is the value $\beta = -2.5$ performing better than $\beta = -2.75$. Here the same might hold for the (1+1) EA. The number of repetitions of the algorithm might simply be too small to make the small difference in the performance between the two values visible. The difference in the performance for the $pmut_\beta$ operator is not as drastic as for the (1+1) EA. Only $\beta = -1.25$ performs significantly worse the next best value.

5.4.4. Comparison of the best variants

algo type	RLS-R	pmut	EA-SM
algo param	r=2	-3.25	2/n
avg mut/change	1.483	1.693	2.258
avg mut/step	1.500	1.729	2.001
total avg count	2,407	2,695	3,421
avg eval count	2,407	2,695	3,421
max eval count	23,155	57,661	52,762
min eval count	19	11	19
fails	0	0	0
fail ratio	0.000	0.000	0.000
avg fail dif	-	-	-

For the geometric distribution once again a variant of the RLS performs the best. The $pmut_{-3.25}$ performs almost equally good with only the (1+1) EA variant performing clearly worse. Here the algorithms are sorted again by their average number of flips per steps. The theory of seems to be true for this kind of input.
To further confirm the best choice for this kind of input there was another experiment of the best variants. The setup is mostly the same except for having a fixed time limit of 100,000 instead of using $100 \cdot n \ln(n)$ as the limit. The smaller inputs are harder relative to their input size so using $100 \cdot n \ln(n)$ as a bound is too small. The first try was executed with 50,000 as the step limit but there the algorithms performed to bad for $n = 20$. Therefore for the second attempt the step limit was increased to 100,000. The first table lists the number of runs where the different algorithms did not find the optimal solution within the time limit.

input size	20	50	100	500	1000	5000	10000
RLS	983	951	885	619	425	42	0
RLS-R (2)	881	578	171	0	0	0	0
(1+1) EA (1/n)	464	132	48	1	1	0	0
(1+1) EA (2/n)	149	11	1	0	0	0	0
pmut(-3.25)	284	67	27	0	0	0	0
pmut(-3.5)	312	91	38	0	1	0	0

For small inputs the geometric distributed input seems to be likely to not have a perfect partition or only very few because there were many iterations where neither of the algorithms found an optimal solution within the time limit. It seems many of the algorithms especially the variants of the RLS seem to be likely to get stuck in a local optima. The (1+1) finds an optimum in most of the runs, so the geometric distributed inputs also seem to be likely to have a perfect partition for small values. They are definitely not as solvable as the binomial inputs but they still have a perfect partition most times. The next table visualises the average number of iterations the algorithm needed for finding an optimal solution if the algorithm managed to do so.

input size	20	50	100	500	1000	5000	10000
RLS	35	78	140	566	904	2119	2188
RLS-R (2)	357	2024	5369	4687	3945	2752	2583
(1+1) EA (1/n)	21827	20775	14583	9459	7702	4358	3924
(1+1) EA (2/n)	18211	11613	7529	5266	4359	3858	3293
pmut(-3.25)	22530	16421	11312	5909	5375	2843	2246
pmut(-3.5)	24202	17414	11731	6503	5216	2773	2388

The variants of the (1+1) EA and of the *pmut* algorithm seem to take about 20,000 iterations for $n = 20$ if they manage to find the optimal solution. They also perform better and better the lager the input gets. This is probability caused by the many additional small values that can be used for smaller adjustments to the fitness. Also a really high value does not have as much of an input, because there are possibly other larger values which cancel each other out, if they are in different bins. The last table again lists the total average number of steps.

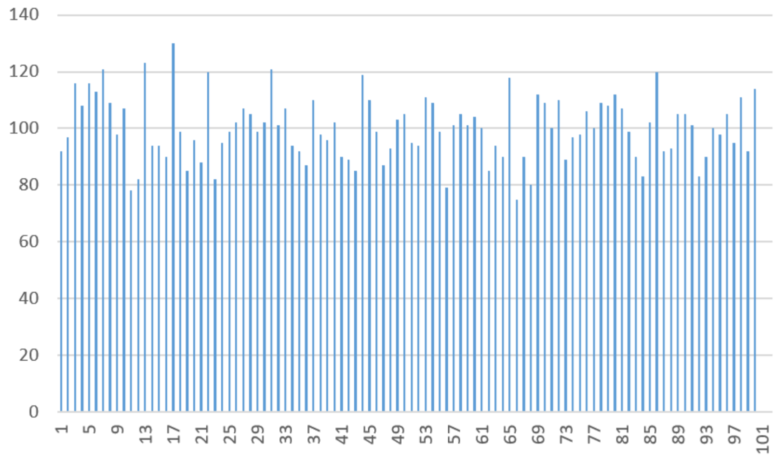
input size	20	50	100	500	1000	5000	10000
RLS	98300	95103	88516	62115	43019	6230	2188
RLS-R (2)	88142	58654	21551	4687	3945	2752	2583
(1+1) EA (1/n)	58099	31232	18683	9550	7795	4358	3924
(1+1) EA (2/n)	30397	12585	7622	5266	4359	3858	3293
pmut(-3.25)	44531	22021	13707	5909	5375	2843	2246
pmut(-3.5)	47851	24929	15086	6503	5311	2773	2388

The RLS is only an option if the input is large enough $n \geq 10,000$. For smaller input sizes especially for $n \leq 100$ choosing the (1+1) EA with mutation rate $2/n$ seems like the best choice. For larger values this (1+1) EA does not find an optimal solution the fastest but is still fast enough to be a viable option. Another rather save option is $pmut\_3.25$. This algorithm performs worse for $n \leq 100$ but is still good in comparison to the other algorithms. For $n \geq 1000 pmut\_3.25$ start to outperform the best version of the (1+1) EA and almost all other researched algorithms.

5.5. Uniform distributed inputs

For the uniform distribution the default values were 1 for the lower bound and 50000 for the upper bound (exclusive). The range was limited to 50000 to reduce the time the algorithms needs to find an optimal solution. The higher the values are with too few values the more likely the input is to not have a perfect partition[BCP01]. This will cause the algorithms to always reach the limit for the number of iterations which drastically increases the time needed for the experiment. The length of the input was 50000.

Figure 5.14.: Distribution of a random uniform input (10000 values between 1 and 100)



5.5.1. RLS Comparison

algo type	RLS-N	RLS-R	RLS-R	RLS-N	RLS-R	RLS-N	RLS
algo param	n=2	r=3	r=4	n=3	r=2	n=4	-
avg mut/change	2.000	1.996	2.476	3.000	1.502	4.000	1.000
avg mut/step	2.000	2.000	2.500	3.000	1.500	4.000	1.000
total avg count	83,118	104,748	105,513	112,223	114,486	121,927	2,443,567
avg eval count	83,118	104,748	105,513	112,223	114,486	121,927	45,834
max eval count	778,110	1,453,252	898,974	1,377,471	915,268	816,633	485,275
min eval count	197	126	45	212	271	155	128
fails	0	0	0	0	0	0	447
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.447
avg fail dif	-	-	-	-	-	-	1

The picture for the RLS variants on this type of input is not clear. There in no obvious tendency for neither of the variants. The only obvious thing is the RLS being the worst of the RLS variants again. Every variant reaches the optimal solution in every case except for the RLS which only manages for roughly 50 % of the inputs. The RLS-N(2) seems to be the best variant for these kinds of inputs. The next best variants are the RLS-(R) with $k = 3$ and $k = 4$ which only differ by 1 %.

5.5.2. (1+1) EA Comparison

algo type	EA-SM	EA-SM	EA-SM	EA-SM	EA-SM	EA
algo param	3/n	2/n	4/n	5/n	10/n	-
avg mut/change	3.102	2.287	4.014	4.937	9.924	1.577
avg mut/step	3.000	2.000	4.000	5.000	10.000	1.000
total avg count	122,098	122,690	124,634	132,509	183,213	213,186
avg eval count	122,098	122,690	124,634	132,509	183,213	213,186
max eval count	956,375	920,658	1,128,158	1,457,069	1,298,089	2,509,163
min eval count	174	188	265	384	6	111
fails	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-

The (1+1) EA seems to perform better with a lower mutation rate. The vales $p_m = 2/n$ and $p_m = 3/n$ reach an optimal solution equally fast. From then on speed of convergence decreases with increasing mutation rate. The only exception from this case is the (1+1) EA which performs the worst despite having the lowest mutation rate. For the uniform distributed input all variants of the (1+1) EA reach an optimal solution within the step limit as for the previous input types.

5.5.3. pmut Comparison

algo type	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut
algo param	-2.50	-2.00	-2.25	-2.75	-1.75	-3.00	-1.50	-3.25	-1.25
avg mut/change	2.866	8.980	4.204	2.205	27.674	1.933	102.803	1.720	312.822
avg mut/step	2.932	10.108	4.559	2.274	34.643	1.934	158.163	1.729	719.965
total avg count	117,346	121,090	121,818	126,467	128,188	140,882	142,970	150,311	193,296
avg eval count	117,346	121,090	121,818	126,467	128,188	140,882	142,970	150,311	193,296
max eval count	1,655,807	1,421,071	1,427,930	2,490,695	2,127,979	1,670,194	1,565,473	1,382,253	1,523,513
min eval count	61	186	76	130	357	155	113	226	13
fails	0	0	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-	-	-

The optimal value for β seems to be somewhere around -2.0 to -2.5, but every other parameter is almost as good as the best. The values next to this interval start to decrease in both directions. The values equally wide apart from -2.25 perform equally good.

5.5.4. Comparison of the best variants

algo type	RLS-N	EA-SM	pmut
algo param	n=2	3/n	-2.25
avg mut/change	2.000	3.109	4.273
avg mut/step	2.000	3.000	4.555
total avg count	84,884	116,576	124,046
avg eval count	84,884	116,576	124,046
max eval count	741,833	1,176,762	1,159,541
min eval count	52	178	178
fails	0	0	0
fail ratio	0.000	0.000	0.000
avg fail dif	-	-	-

For the uniform distributed input the best variant of the RLS once again seems to be the best algorithm. But by looking at the smaller values again this does not hold in general.

input size	20	50	100	500	1000	5000	10000
RLS-N(2)	987	983	884	382	324	291	266
RLS-R (3)	975	755	642	465	439	400	422
RLS-R (4)	905	637	548	482	438	408	415
(1+1) EA (2/n)	858	730	659	496	483	496	468
(1+1) EA (3/n)	762	575	535	451	433	405	411
(1+1) EA (4/n)	664	523	496	428	408	424	419
pmut(-2.5)	854	772	668	548	517	488	462

The RLS variants are the most likely to get stuck in a local optimum for $n \leq 100$. The (1+1) EA variants also often do not find an optimal solution, but this happens less frequently. The more values the input has the more likely it is for any of the algorithms to find a perfect partition. Between $n = 100$ and $n = 500$ the performance of the RLS-N(2) drastically increases and for $n \geq 500$ this variant of the RLS stays the best variant for the remaining input sizes.

input size	20	50	100	500	1000	5000	10000
RLS-N(2)	252	1382	6378	34091	36030	38409	39365
RLS-R (3)	4012	33662	35519	40678	39966	38438	35784
RLS-R (4)	18663	39434	39068	41588	39252	39578	38932
(1+1) EA (2/n)	31595	39959	38253	40912	41287	38213	38151
(1+1) EA (3/n)	32990	41302	39362	40242	39432	41066	40864
(1+1) EA (4/n)	33453	41598	38060	41584	39748	39617	41490
pmut(-2.5)	38488	41365	39010	42765	44430	38055	37215

The steps needed to find an optimal solution seems to be nearly constant for every algorithm as the number of steps does not strictly increase with n but sometimes even decreases. The decreases are most likely caused by fluctuations.

input size	20	50	100	500	1000	5000	10000
RLS-N(2)	98703	98323	89139	59268	56756	56332	55494
RLS-R (3)	97600	83747	76916	68263	66320	63062	62883
RLS-R (4)	92273	78014	72458	69742	65859	64230	64275
(1+1) EA (2/n)	90286	83789	78944	70220	69645	68859	67096
(1+1) EA (3/n)	84051	75053	71803	67193	65658	64934	65169
(1+1) EA (4/n)	77640	72142	68782	66586	64330	65219	66005
pmut(-2.5)	91019	86631	79751	74129	73159	68284	66221

My general advice would be choosing the RLS-N(2) for $n \geq 500$ and the (1+1) EA with $p_m = 4/n$ otherwise.

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 interval. 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 $\mathcal{O}(nlogn)$ 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 $\mathcal{O}(nlogn)$

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

5.6.1. RLS Comparison

algo type	RLS	RLS-R	RLS-R	RLS-R	RLS-N	RLS-N	RLS-N
algo param	-	r=2	r=3	r=4	n=2	n=3	n=4
avg mut/change	1.000	1.181	1.688	1.865	1.997	3.000	3.997
avg mut/step	1.000	1.500	2.000	2.500	2.000	3.000	4.000
total avg count	90,931	168,311	236,317	307,533	921,030	921,030	921,030
avg eval count	90,931	168,311	236,317	307,533	-	-	-
max eval count	156,854	296,206	498,474	595,831	0	0	0
min eval count	64,941	120,582	158,304	212,193	-	-	-
fails	0	0	0	0	1,000	1,000	1,000
fail ratio	0.000	0.000	0.000	0.000	1.000	1.000	1.000
avg fail dif	-	-	-	-	53	36	263

5.6.2. (1+1) EA Comparison

algo type	EA	EA-SM	EA-SM	EA-SM	EA-SM	EA-SM	EA-SM	EA-SM
algo param	-	2/n	3/n	4/n	5/n	10/n	50/n	100/n
avg mut/change	1.273	1.750	2.334	2.965	3.636	7.288	45.923	95.590
avg mut/step	1.000	2.000	3.000	4.000	5.000	10.000	49.998	100.00
total avg count	230,328	297,602	495,951	860,736	921,030	921,030	921,030	921,030
avg eval count	230,328	297,602	495,951	812,983	-	-	-	-
max eval count	399,393	625,976	839,325	917,029	-	-	-	-
min eval count	162,400	193,796	347,185	635,812	-	-	-	-
fails	0	0	0	99	224	224	224	224
fail ratio	0.000	0.000	0.000	0.442	1.000	1.000	1.000	1.000
avg fail dif	-	-	-	1	18	570	2,488	3,115

5.6.3. pmut Comparison

algo type	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut
algo param	-3.25	-3.00	-2.75	-2.50	-2.25	-2.00	-1.75	-1.50	-1.25
avg mut/change	1.289	1.359	1.459	1.591	1.813	2.207	2.760	3.604	5.382
avg mut/step	1.731	1.934	2.270	2.907	4.371	8.486	22.299	70.692	224.466
total avg count	145,095	149,664	170,912	181,099	214,361	249,102	301,566	415,413	715,219
avg eval count	145,095	149,664	170,912	181,099	214,361	249,102	301,566	415,413	683,204
max eval count	217,932	223,561	254,330	246,635	365,378	376,768	431,629	735,214	853,181
min eval count	111,061	119,931	120,965	130,174	161,244	180,570	232,166	311,979	492,686
fails	0	0	0	0	0	0	0	0	7
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.135
avg fail dif	-	-	-	-	-	-	-	-	1

5.6.4. Comparison of the best variants

algo type	RLS	pmut	EA
algo param	-	-3.25	-
avg mut/change	1.000	1.287	1.272
avg mut/step	1.000	1.729	1.000
total avg count	91,171	143,121	231,082
avg eval count	91,171	143,121	231,082
max eval count	153,143	227,737	446,942
min eval count	65,783	93,602	165,818
fails	0	0	0
fail ratio	0.000	0.000	0.000
avg fail dif	-	-	-

5.7. Carsten Witts worst case input

5.7.1. RLS Comparison

algo type	RLS-N	RLS-R	RLS-R	RLS-R	RLS-N	RLS	RLS-N
algo param	n=3	r=4	r=3	r=2	n=4	-	n=2
avg mut/change	3.000	2.379	1.985	1.327	3.997	1.000	1.998
avg mut/step	3.000	2.500	2.000	1.500	4.000	1.000	2.000
total avg count	1,436	1,641	1,905	2,856	4,420	6,507	6,693
avg eval count	1,436	1,641	1,905	2,856	4,420	3,755	6,693
max eval count	12,521	15,338	22,394	26,297	42,702	31,837	74,281
min eval count	0	0	0	0	0	0	0
fails	0	0	0	0	0	3	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.003	0.000
avg fail dif	-	-	-	-	-	4,249	-

5.7.2. (1+1) EA Comparison

algo type	EA-SM	EA-SM	EA-SM	EA-SM	EA-SM	EA-SM	EA-SM	EA
algo param	100/n	50/n	10/n	5/n	4/n	3/n	2/n	-
avg mut/change	99.982	49.983	10.028	5.085	4.112	3.150	2.244	1.470
avg mut/step	99.980	49.975	10.002	5.001	4.001	3.000	2.001	1.000
total avg count	73	97	397	839	966	1,391	1,827	3,732
avg eval count	73	97	397	839	966	1,391	1,827	3,732
max eval count	488	736	5,075	8,348	9,734	14,546	22,186	44,370
min eval count	0	0	0	0	0	0	0	0
fails	0	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-	-

5.7.3. pmut Comparison

algo type	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut
algo param	-1.25	-1.50	-1.75	-2.00	-2.25	-2.50	-2.75	-3.00	-3.25
avg mut/change	197.675	69.384	23.211	8.999	4.259	2.819	2.133	1.802	1.598
avg mut/step	226.848	69.933	22.429	8.747	4.313	2.911	2.268	1.934	1.725
total avg count	41	85	222	536	922	1,368	1,723	2,080	2,339
avg eval count	41	85	222	536	922	1,368	1,723	2,080	2,339
max eval count	226	682	2,033	4,760	9,749	16,271	18,953	18,794	25,383
min eval count	0	0	0	0	0	0	0	0	0
fails	0	0	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-	-	-

5.7.4. Comparison of the best variants

algo type	pmut	EA-SM	RLS-N
algo param	-1.25	100/n	k=3
avg mut/change	208.477	99.891	3.000
avg mut/step	228.998	100.014	3.000
total avg count	42	68	1,216
avg eval count	42	68	1,216
max eval count	231	466	13,896
min eval count	0	0	0
fails	0	0	0
fail ratio	0.000	0.000	0.000
avg fail dif	-	-	-

5.8. Multiple distributions overlapped

5.8.1. RLS Comparison

algo type	RLS-R	RLS-R	RLS-R	RLS-N	RLS-N	RLS	RLS-N
algo param	r=4	r=3	r=2	n=3	n=2	-	n=4
avg mut/change	2.413	1.951	1.478	2.000	1.000	1.000	3.000
avg mut/step	2.503	2.000	1.500	2.000	1.000	1.000	3.000
total avg count	539	546	608	612	729	733	930
avg eval count	539	546	608	612	729	733	930
max eval count	1,579	1,661	1,883	2,629	2,623	2,371	4,591
min eval count	15	8	42	53	80	59	73
fails	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-

5.8.2. (1+1) EA Comparison

algo type	EA-SM	EA-SM	EA-SM	EA	EA-SM	EA-SM
algo param	3/n	2/n	4/n	-	5/n	10/n
avg mut/change	3.009	2.230	3.863	1.545	4.762	9.548
avg mut/step	3.000	2.000	3.998	0.999	4.998	10.000
total avg count	599	617	701	942	968	11,682
avg eval count	599	617	701	942	968	11,682
max eval count	1,927	1,769	2,016	3,284	3,537	63,180
min eval count	56	65	85	91	51	112
fails	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-

5.8.3. pmut Comparison

algo type	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut
algo param	-1.75	-2.00	-2.25	-1.50	-2.50	-2.75	-3.00	-3.25	-1.25
avg mut/change	29.356	9.267	3.910	123.650	2.761	2.150	1.855	1.673	399.522
avg mut/step	40.906	10.981	4.619	226.724	2.950	2.256	1.935	1.729	1192.167
total avg count	454	479	501	504	515	541	555	572	722
avg eval count	454	479	501	504	515	541	555	572	722
max eval count	1,404	1,423	1,437	1,606	1,988	1,434	1,444	1,800	2,223
min eval count	38	19	45	47	18	18	58	27	48
fails	0	0	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-	-	-

5.8.4. Comparison of the best variants

algo type	pmut	RLS-R	EA-SM
algo param	-1.75	r=4	3/n
avg mut/change	30.999	2.415	3.015
avg mut/step	42.499	2.501	3.002
total avg count	463	543	586
avg eval count	463	543	586
max eval count	1,330	1,613	1,389
min eval count	61	50	99
fails	0	0	0
fail ratio	0.000	0.000	0.000
avg fail dif	-	-	-

5.9. Multiple distributions mixed

5.9.1. RLS Comparison

algo type	RLS-N	RLS-R	RLS-R	RLS-R	RLS-N	RLS-N	RLS
algo param	n=3	r=4	r=3	r=2	n=4	n=2	-
avg mut/change	2.000	2.556	2.059	1.593	3.000	NaN	NaN
avg mut/step	2.000	2.501	2.000	1.500	3.000	NaN	NaN
total avg count	125,951	249,157	265,643	268,552	307,558	9,210,300	9,210,300
avg eval count	125,951	249,157	265,643	268,552	307,558	-	-
max eval count	862,008	1,939,594	2,389,907	2,012,463	2,203,167	-	-
min eval count	155	276	331	267	156	-	-
fails	0	0	0	0	0	1,000	1,000
fail ratio	0.000	0.000	0.000	0.000	0.000	1.000	1.000
avg fail dif	-	-	-	-	-	778	785

5.9.2. (1+1) EA Comparison

algo type	EA-SM	EA-SM	EA-SM	EA-SM	EA-SM	EA
algo param	4/n	3/n	5/n	2/n	10/n	-
avg mut/change	3.981	3.153	4.887	2.359	9.759	1.701
avg mut/step	4.000	3.000	5.000	2.000	10.000	1.000
total avg count	244,704	253,722	268,529	304,147	346,669	577,955
avg eval count	244,704	253,722	268,529	304,147	346,669	577,955
max eval count	2,209,416	2,340,925	2,589,980	2,416,509	2,474,967	3,776,445
min eval count	269	405	24	177	236	89
fails	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-

5.9.3. pmut Comparison

algo type	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut
algo param	-2.00	-2.25	-1.75	-2.50	-2.75	-1.50	-3.00	-3.25	-1.25
avg mut/change	6.738	4.132	14.905	2.817	2.298	37.736	2.008	1.832	96.985
avg mut/step	8.485	4.364	22.232	2.906	2.271	70.714	1.934	1.729	224.557
total avg count	292,521	292,763	304,924	307,403	322,391	337,925	356,686	377,068	419,335
avg eval count	292,521	292,763	304,924	307,403	322,391	337,925	356,686	377,068	419,335
max eval count	3,811,952	1,930,058	2,875,275	2,364,678	2,233,600	4,824,371	2,832,218	3,395,371	3,133,351
min eval count	535	251	202	105	124	560	104	52	841
fails	0	0	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-	-	-

5.9.4. Comparison of the best variants

algo type	RLS-N	EA-SM	pmut
algo param	n=3	4/n	-2.00
avg mut/change	2.000	3.996	7.729
avg mut/step	2.000	4.000	8.477
total avg count	129,281	267,444	311,741
avg eval count	129,281	267,444	311,741
max eval count	1,276,682	1,958,436	2,465,518
min eval count	62	385	245
fails	0	0	0
fail ratio	0.000	0.000	0.000
avg fail dif	-	-	-

5.10. Multiple distributions mixed & overlapped

5.10.1. RLS Comparison

algo type	RLS-N	RLS-R	RLS-R	RLS-R	RLS	RLS-N	RLS-N
algo param	n=3	r=3	r=2	r=4	-	n=2	n=4
avg mut/change	2.000	1.903	1.468	2.321	1.000	1.000	3.000
avg mut/step	2.000	2.001	1.500	2.500	1.000	1.000	3.000
total avg count	1,193	1,326	1,348	1,406	1,776	1,780	1,850
avg eval count	1,193	1,326	1,348	1,406	1,776	1,780	1,850
max eval count	3,692	3,666	3,490	3,833	4,568	4,700	5,159
min eval count	44	65	94	33	109	55	102
fails	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-

5.10.2. (1+1) EA Comparison

algo type	EA-SM	EA-SM	EA-SM	EA	EA-SM	EA-SM
algo param	3/n	2/n	4/n	-	5/n	10/n
avg mut/change	2.883	2.146	3.698	1.514	4.583	9.553
avg mut/step	3.000	2.002	4.000	1.001	5.001	10.000
total avg count	1,577	1,648	1,938	2,224	2,579	22,183
avg eval count	1,577	1,648	1,938	2,224	2,579	22,183
max eval count	4,466	4,510	5,481	6,566	8,345	93,948
min eval count	66	169	159	232	179	225
fails	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-

5.10.3. pmut Comparison

algo type	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut	pmut
algo param	-2.50	-2.25	-2.75	-2.00	-3.25	-3.00	-1.75	-1.50	-1.25
avg mut/change	2.500	3.777	2.053	7.773	1.622	1.785	22.640	97.028	310.342
avg mut/step	2.914	4.680	2.271	10.908	1.733	1.928	41.606	222.441	1196.549
total avg count	1,368	1,395	1,418	1,422	1,423	1,428	1,554	1,797	2,493
avg eval count	1,368	1,395	1,418	1,422	1,423	1,428	1,554	1,797	2,493
max eval count	4,195	4,235	4,599	4,348	4,609	4,040	4,775	5,609	8,303
min eval count	133	90	49	23	95	99	94	134	87
fails	0	0	0	0	0	0	0	0	0
fail ratio	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
avg fail dif	-	-	-	-	-	-	-	-	-

5.10.4. Comparison of the best variants

algo type	RLS-N	pmut	EA-SM
algo param	n=3	-2.50	3/n
avg mut/change	2.000	2.875	2.882
avg mut/step	2.000	2.972	3.000
total avg count	1,225	1,424	1,579
avg eval count	1,225	1,424	1,579
max eval count	3,347	3,791	4,611
min eval count	50	139	103
fails	0	0	0
fail ratio	0.000	0.000	0.000
avg fail dif	-	-	-

5.10.5. Conclusion of empirical results

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