Generalised Jump Functions

ThRaSH Seminar Series

Henry Bambury, Antoine Bultel
Joint work with Prof. Benjamin Doerr for GECCO2021

École polytechnique

June 3, 2021

Contents

Motivation: Multimodal Benchmark Functions for Evolutionary Algorithms

Framework of study

- **Pseudo-Boolean optimization:** $f : \{0,1\}^n \to \mathbb{R}$, find (one of) its global maximum.

- **A wide class of algorithms:** Evolutionary Algorithms.
Algorithms that rely on notions of mutation and selection for optimization purposes.

- $\{0,1\}^n$ is the **searchspace**;
x is an **individual**;
f is the **fitness function**.

Benchmark functions

It is standard to focus on a few representative functions to gauge strengths and weaknesses. Famous ones:

- OneMax
- LeadingOnes
- Jump_k
- *etc.*

The choice and design of benchmark functions is a cornerstone of theory of EAs.
What should be expected from a good benchmark function? A very good debate to have!

Unimodalness and multimodalness

A pseudo-Boolean function is said to be **unimodal** if it has at most one local maximum, **multimodal** otherwise.

- Unimodal functions are quite rare among all p-B functions; likewise, in real-life, optimization problems without local optima are not very common.
- Crucial need to study and understand how Randomized Search Heuristics deal with local optima.

Multimodal benchmark functions?

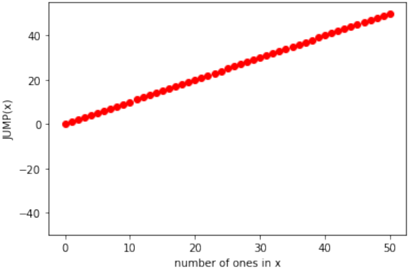
Yet, the vast majority of benchmark functions are unimodal.

- OneMax
- LeadingOnes
- *etc.*

The only standard widely used multimodal benchmark functions are the $\textsc{Jump}_k$ functions.

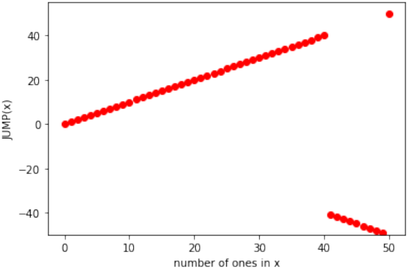
Examples

$$\textsc{OneMax}(x) = \|x\|_1$$



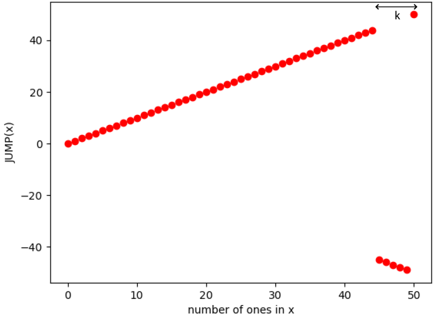
Unimodal

$$\textsc{Jump}_k(x) = \begin{cases} \|x\|_1 & \text{if } \|x\|_1 \in [0..n-k] \cup \{n\}, \\ -\|x\|_1 & \text{otherwise,} \end{cases}$$



Multimodal

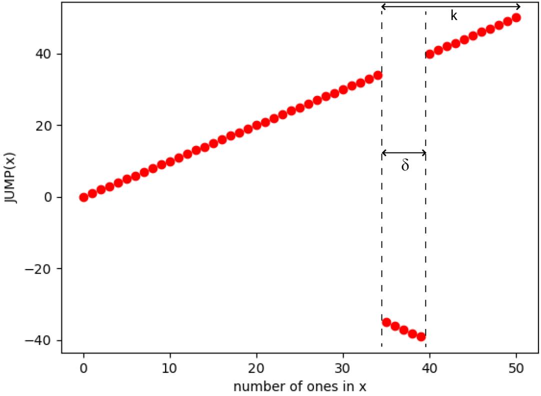
The JUMP_k Functions



- A layer of local optima, at Hamming distance k from the global optimum.
- Fair evaluation of the ability of an algorithm to leave a local optimum.
- One flaw: when stuck, the only way to leave local optima is a *perfect jump*. This is quite a specific feature!

$\text{JUMP}_{k,\delta}$: a Generalized JUMP_k

$$\text{JUMP}_{k,\delta}(x) = \begin{cases} \|x\|_1 & \text{if } \|x\|_1 \in [0..n-k] \cup [n-k+\delta..n], \\ -\|x\|_1 & \text{otherwise.} \end{cases}$$



We also introduce $\ell = k - \delta$.

Consequences of this generalization

Contrarily to what one could think, $\textsc{Jump}_{k,\delta}$ is **not** equivalent to $\textsc{Jump}_{\delta}$ followed by $\textsc{OneMax}$.

- On $\textsc{Jump}_5$ with $n = 40$, when stuck on the local optima, there is only 1 point with strictly better fitness.
- On $\textsc{Jump}_{10,5}$ with $n = 40$, there are $\sum_{i=1,2,3,4,5} \binom{10}{i} = 647$.

Intuition

Since crossing the valley is (way) easier on $\textsc{Jump}_{k,\delta}$, algorithms should benefit from an exponential speedup.

Objectives

We focused on several EAs, whose runtimes on $\textsc{Jump}_k$ were determined in previous research. We study their performance on $\textsc{Jump}_{k,\delta}$.

We hope to show non-trivial phenomenons. If such phenomenons appear, they will prove the genuine interest of generalizing the $\textsc{Jump}$ functions.

Impact on the runtime of algorithms

1. $(1+1)$ **EA with Fixed Mutation Rate**
2. Fast $(1+1)$ EA
3. RLS with Stagnation Detection
4. $(1+1)$ EA with Stagnation Detection

Algorithm 1: The $(1+1)$ EA with fitness function $f : \{0,1\}^n \to \mathbb{R}$ and static mutation rate p

Initialization;
$x \in \{0,1\}^n \leftarrow$ uniform at random;
Optimization;
repeat
  Sample $y \in \{0,1\}^n$ by flipping each bit in x with probability p;
  if $f(y) \geq f(x)$ **then**
    $x \leftarrow y$
until *Stopping condition*;

Performance on $\textsc{Jump}_k$

Result obtained by Doerr et al. in [DLMN17].

$(1+1)$ EA on $\textsc{Jump}_k$

The best possible expected optimization time of the $(1+1)$ EA on $\textsc{Jump}_k$ is asymptotically achieved in $p = \frac{k}{n}$, and is asymptotically

$$\Theta\left(\left(\frac{k}{n}\right)^{-k}\left(\frac{n}{n-k}\right)^{n-k}\right).$$

Furthermore, any deviation from that value leads to exponential loss in runtime.

It is not obvious whether this generalizes to $\textsc{Jump}_{k,\delta}$.

Performance on $\text{JUMP}_{k,\delta}$

General bounds

For all $k, \ell, n \in \mathbb{N}$ such that $k \leq \frac{n}{2}$ and all $p \leq \frac{1}{2}$, let $T_p(k, \ell, n)$ be the expected optimization time of the $(1+1)$ EA with fixed mutation rate p on the $\text{JUMP}_{k,\ell,n}$ problem. Then

$$\frac{1}{2^n}\sum_{i=0}^{n-k}\binom{n}{i}\frac{1}{F(p)} \leq T_p(k,\ell,n) \leq \frac{1}{F(p)} + \frac{\ln(n)+1}{p(1-p)^{n-1}}.$$

Where

$$F(p) := \sum_{j=0}^{\ell}\sum_{i=0}^{n-k}\binom{k}{k-\ell+i+j}\binom{n-k}{i}p^{k-\ell+2i+j}(1-p)^{n-k+\ell-2i-j}$$

is the probability of jumping over the valley from a local optimum.

Ideas of proof

Formula for $F(p)$

- The probability of the event "Jumping over the valley from the local optimum".
- We must flip at least δ bad bits.
- Enumerate all scenarios.

Lower Bound

- If the initial searchpoint is before the valley, the runtime stochastically dominates a variable geometric law of parameter $F(p)$.

Upper Bound

- Define fitness layers.
- A run of the algorithm is a random walk between layers.
- Conclude using the fitness level theorem [Weg01].

$F(p)$

$$F(p) := \sum_{j=0}^{\ell} \sum_{i=0}^{n-k} \binom{k}{k-\ell+i+j} \binom{n-k}{i} p^{k-\ell+2i+j} (1-p)^{n-k+\ell-2i-j}$$

plays a crucial role in the phenomena we study.

- On $\textsc{Jump}_k$, $F(p) = p^k(1-p)^{(n-k)}$.
- Questions that were simple on $\textsc{Jump}_k$ are now highly non-trivial :
    - What are the maxima of $F(p)$?
    - For fixed p, can we have a simple asymptotic equivalent for $F(p)$?

Standard regime

The aforementioned bounds are very hard to handle without controlling k, δ. To continue, we had to define a reasonable regime.

Definition (Standard regime)

Let the *standard regime* (SR) be the space in which: $k = o(n^{1/3})$

Motivation:

Standard regime

In the SR, if furthermore $p = o(\frac{1}{\sqrt{n\ell}})$, then $F(p) = (1 + o(1))\binom{k}{\delta} p^{\delta}(1-p)^{n-\delta}$.

Sketch of proof

Direct asymptotic calculations. Nasty but not straightforward : at several points, combinatorial tricks are required.

Standard regime

Performance in the standard regime

In the SR, if furthermore $p = o(\frac{1}{\sqrt{n\ell}})$,

$$T_p(k, \delta, n) = (1 + o(1)) \frac{1}{\binom{k}{\delta} p^\delta (1-p)^{n-\delta}}.$$

Theorem

In the SR, the asymptotic best choice of p is $p = \frac{\delta}{n}$, which gives the runtime

$$T_{\delta/n}(k, \delta, n) = (1 + o(1)) \binom{k}{\delta}^{-1} \left(\frac{en}{\delta}\right)^\delta,$$

and any deviation from that optimal value results in exponential in δ loss on the runtime.

Sketch of proof

Formula for $T_p(k, \delta, n)$

Direct calculations.

Optimality of $p = \delta/n$

- Key point: show that $F(p)$ is decreasing on $[\frac{k+\ell}{n}, +\infty[$, and notice $\frac{k+\ell}{n} = o(\frac{1}{\sqrt{n\ell}})$ in the SR.
- So the optimal p can't be in $[\frac{k+\ell}{n}, +\infty[$.
- for all other p, we have the formula for $T_p(k, \delta, n)$, it is obviously minimal on $p = \frac{\delta}{n}$.

Exponential loss for other p

Direct (nasty) calculations

Discussion

- Extends perfectly what was known on $\textsc{Jump}_k$ from [DLMN17], but only in the SR.
- The SR is quite large and covers many interesting settings.
- But it is **not** the only interesting regime! *(There is no particular reason to restrict k, why not considering settings with $k = \Theta(n)$?)*
- We did not find tools to study simply those other regimes. An interesting direction for future work!

1. $(1+1)$ EA with Fixed Mutation Rate
2. **Fast** $(1+1)$ **EA**
3. RLS with Stagnation Detection
4. $(1+1)$ EA with Stagnation Detection

$(1+1)$ FEA$_\beta$

- Introduced in [DLMN17] to improve the simple $(1+1)$ EA on $\textsc{Jump}_k$.
- The mutation rate is chosen randomly at each iteration, using a power-law distribution.

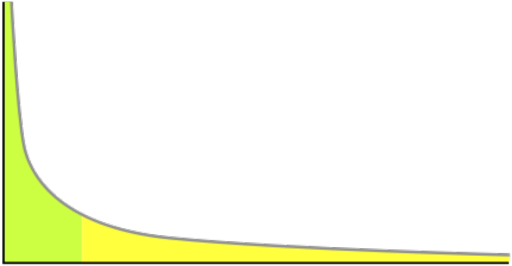


Figure: Plot of a Heavy-tailed power-law distribution.

Pseudocode

Algorithm 2: The $(1+1)$ FEA$_\beta$ with fitness $f:\{0,1\}^n \to \mathbb{R}$

```
Initialization;
x ∈ {0,1}^n ← uniform at random;
Optimization;
repeat
    Sample α randomly in [1..n/2] with power-law distribution D^β_{n/2};
    Sample y ∈ {0,1}^n by flipping each bit in x with probability α/n;
    if f(y) ≥ f(x) then
        x ← y
until Stopping condition;
```

Performance on JUMP_k

The runtime of $(1+1)$ FEA_β is a **small polynomial above** the best runtime with fixed mutation rate.

Theorem [DLMN17]

Let $n \in \mathbb{N}$ and $\beta > 1$. For all $k \in [2..n/2]$, with $m > \beta - 1$, the expected optimization time $T_\beta(k, n)$ of the $(1+1)$ FEA_β on JUMP_k satisfies

$$T_\beta(k, n) = O\left(C^{\beta}_{n/2} k^{\beta - 0.5} T_{opt}(k, n)\right),$$

Where $T_{opt}(k, n)$ is the expected runtime of the simple $(1+1)$ EA with the optimal fixed mutation rate $p = \frac{k}{n}$.

Performance on $\textsc{Jump}_{k,\delta}$

We proved that the result generalizes well.

Theorem

Let $n \in \mathbb{N}$ and $\beta > 1$. For all k, δ in the standard regime, with $\delta > \beta - 1$, the expected optimization time $T_\beta(k, \delta, n)$ of the $(1+1)$ FEA_β satisfies

$$T_\beta(k, \delta, n) = O\left(C_{n/2}^{\beta} \delta^{\beta - 0.5} T_{\delta/n}(k, \delta, n)\right).$$

Sketch of proof

Exactly like in [DLMN17]. The fitness level theorem, along with a decent amount of asymptotic computations, directly gives the result.

1. $(1+1)$ EA with Fixed Mutation Rate
2. Fast $(1+1)$ EA
3. **RLS with Stagnation Detection**
4. $(1+1)$ EA with Stagnation Detection

SD-RLS

- Introduced earlier this year by A.Rajabi and C.Witt in [RW21].
- As an improvement SD-$(1+1)$ EA for $\textsc{Jump}_k$ (which we will discuss later).
- For each iteration, instead of *standard bit mutation* with $p = \frac{r}{n}$, *randomized local search* of strength r is used. Exactly r bits are chosen uniformly at random and flipped.
- **The strength changes along the run, but not randomly:**
  - Initialized as $r = 1$.
  - If the algorithm stays stuck in the same layer for $\ln(R)\binom{n}{r}$ iterations, then with probability at least $1 - \frac{1}{R}$ there is no improvement at Hamming distance r (R is a control parameter). In this case the strength is increased to $r + 1$.
  - When a strictly better search point is found, return to $r = 1$.

SD-RLS*

Problem: With SD-RLS, termination is not ensured. If the only improvement is at Hamming distance m from the search point, and missed during phase m, the algorithms does not terminate.

Solution: Visit the strengths in a different order.

- SD-RLS: $\mathbf{1} \rightarrow \mathbf{2} \rightarrow \mathbf{3} \rightarrow \mathbf{4} \rightarrow \dots$
- SD-RLS*: $\mathbf{1} \rightarrow \mathbf{2} - 1 \rightarrow \mathbf{3} - 2 - 1 \rightarrow \mathbf{4} - 3 - 2 - 1 \rightarrow \dots$

The full pseudocode can be found in [RW21].

Performance on $\textsc{Jump}_k$

Theorem [RW21]

Let $n \in \mathbb{N}$. Let $T_{SD-RLS^*}(k,n)$ be the expected runtime of the SD-RLS* on $\textsc{Jump}_k$, with $R \geq n^{2+\varepsilon}$ for some constant $\varepsilon > 0$. For all $k \geq 2$,

$$T_{SD-RLS^*}(k,n) = \begin{cases} \binom{n}{k}\left(1 + O\left(\frac{k^2}{n-2k}\ln(n)\right)\right) & \text{if } k < n/2, \\ O(2^n n \ln(n)) & \text{if } k \geq n/2. \end{cases}$$

Better than the $(1+1)$ EA with optimal mutation rate by a factor $\left(\frac{en}{k}\right)^{-k}\binom{n}{k}$, which is at least $1/e$ for small values of k.

Intuitive runtime analysis

Behind this expected runtime, we can see two quantities:

Number of iterations with strength $r < k$

$+$

Number of iterations needed to jump once strength k is reached.

The first are wasted steps, but their number is of the same order as the other steps. All in all, this sacrifice is worth it.

Intuitive runtime analysis

But what happens when we move to $\textsc{Jump}_{k,\delta}$?

Number of iterations with strength $r < \delta$

[Not divided by $\binom{k}{\delta}$]

$+$

Number of iterations needed to jump once strength δ is reached.

[Divided by $\binom{k}{\delta}$]

If $\binom{k}{\delta}$ is large enough, the length of the "wasted steps" is dominant, so the sacrifice becomes costly (and SD-RLS* is slowed down).

Performance on $\textsc{Jump}_{k,\delta}$

Theorem

Let $T_{SD-RLS^*}(k, \delta, n)$ be the runtime of the SD-RLS* on $\textsc{Jump}_{k,\delta}$. Suppose that there exists a constant $\varepsilon > 0$ such that the control parameter is $R \geq n^{2+\varepsilon}$. Then if $k \leq n - \omega(\sqrt{n})$ and $\delta \geq 3$,

$$T_{SD-RLS^*}(k, \delta, n) = (1 + o(1)) \left[\ln(R) \sum_{i=1}^{\delta-1} \sum_{j=0}^{i} \binom{n}{j} + \binom{n}{\delta} \binom{k}{\delta}^{-1} \right].$$

Comparaison to other algorithms

It is easy to see that this runtime is asymptotically larger than the previous ones. The following lemma puts that difference into perspective.

Theorem

For any integer K, there exists an instance of $\text{JUMP}_{k,\delta}$, within the standard regime, on which

$$T_{SD-RLS^*}(k, \delta, n) = \Omega\left(n^{K-1} T_{\frac{1}{n}}(k, \delta, n)\right).$$

1. $(1+1)$ EA with Fixed Mutation Rate
2. Fast $(1+1)$ EA
3. RLS with Stagnation Detection
4. $(1+1)$ **EA with Stagnation Detection**

SD-$(1+1)$ EA

- Introduced in [RW20].
- Same principle as SD-RLS, but standard-bit mutation is used instead of RLS.
- The number of iterations needed to increase the mutation rate from $\frac{r}{n}$ to $\frac{r+1}{n}$ is $2\left(\frac{en}{r}\right)^r \ln(nR)$ instead of $\ln(R)\binom{n}{r}$.

Pseudocode

Algorithm 3: The SD-$(1+1)$ EA with fitness function $f : \{0,1\}^n \to \mathbb{R}$ and parameter R

```
Initialization;
x ∈ {0,1}^n ← uniform at random; u ← 0; r ← 1;
Optimization;
repeat
    Sample y ∈ {0,1}^n by flipping each bit in x with probability r/n;
    u ← u + 1;
    if f(y) > f(x) then
        x ← y; r ← 1; u ← 0;
    else if f(y) = f(x) and r = 1 then
        x ← y;
    if u > 2 (en/r)^r ln(nR) then
        r ← min{r + 1, n/2} ; u ← 0;
until Stopping condition;
```

Performance on $\textsc{Jump}_k$

The SD-$(1+1)$ EA algorithm has a runtime equivalent to the optimal $(1+1)$ EA on $\textsc{Jump}_k$.

Theorem [RW20]

The expected optimization time $T_{SD}(k,n)$ of the SD-$(1+1)$ EA on $\textsc{Jump}_k$ satisfies

$$T_{SD}(k,n) = O\left(\left(\frac{en}{k}\right)^k\right).$$

Intuitive analysis on $\textsc{Jump}_{k,\delta}$

We could conduct the same analysis as for the SD-RLS

Steps with small p, and low probabilty of jumping

$$+$$

Step with larger p, for which the probability of jumping is reasonable

If the first steps last too long, the same phenomenon could happen.
Problem: How do we formalise "small p", and "low probabilty of jumping".

Inefficient steps

Definition (Inefficient steps)

Let $r \in [1..n/2]$. We say that *step r is inefficient* if

$$\left(1 - F\left(\frac{r}{n}\right)\right)^{2\left(\frac{en}{r}\right)^r \ln(nR)} = o(1).$$

Proposition (Bounding the runtime with inefficient steps)

If step r is inefficient, then the expected runtime $T_{SD-OEA}(k, \delta, n)$ of the SD-$(1+1)$ EA on $\textsc{Jump}_{k,\delta}$ satisfies

$$T_{SD-OEA}(k, \delta, n) \geq (1 - o(1))2\left(\frac{en}{r}\right)^r \ln(nR).$$

An interesting compensation phenomenon

Intuitively, in the SR, the runtime of the SD-$(1+1)$ EA should be of about $\frac{1}{F(\frac{\delta}{n})}$.

So we should search for any r such that:

- Step r is inefficient, i.e. $F(\frac{r}{n})2\left(\frac{en}{r}\right)^r \ln(nR)$ is small.
- Its length is not neglectible, i.e. $F\left(\frac{\delta}{n}\right)2\left(\frac{en}{r}\right)^r \ln(nR)$ is big.

Surprisingly enough, **by some compensation phenomenon we do not fully understand**, it seems that such r are very difficult to find, and do not exist in the standard regime.

An example where SD-$(1+1)$ EA suffers exponential loss

We consider the specific instance $k = n/4$, $\delta = n/8$.

Theorem

On this instance of the $\text{JUMP}_{k,\delta}$ problem, the runtimes of the SD-$(1+1)$ EA and of the $(1+1)$ FEA$_\beta$ satisfy

$$T_{SD-OEA}(k,\delta,n) = \Omega\left(e^{\Theta(n)} T_\beta(k,\delta,n)\right).$$

Sketch of proof

Very nasty computations: step $\frac{2n}{38}$ is inefficient. Its length is $\Omega\left(e^{\Theta(n)} T_\beta(k,\delta,n)\right)$.

Discussion

- The SD-$(1+1)$ EA is structurally similar to SD-RLS*, but their behaviour are very different on $\text{JUMP}_{k,\delta}$.
- Only the mutation scheme is changed.
- Yet, this seems to prevent the SD-$(1+1)$ EA from losing too much time on $\text{JUMP}_{k,\delta}$.
- This can be seen experimentally! *(next section)*

We cannot give an explanation for this phenomenon.
We believe that further investigating it could give great understanding of mutation heuristics in general.

Experiments

Experiments

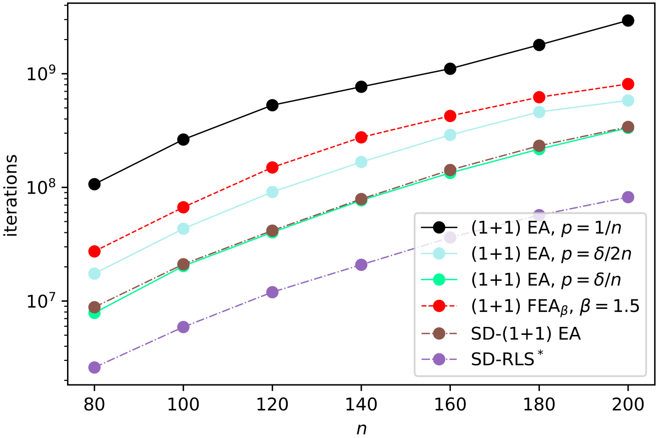


Figure: Optimization times of different algorithms on $\textsc{Jump}_{k,\delta}$ with $k = \delta = 4$.

Experiments

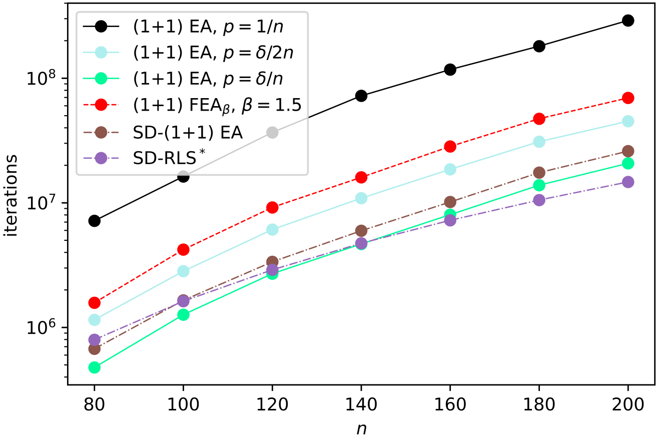


Figure: Optimization times of different algorithms on $\text{JUMP}_{k,\delta}$ with $k = 6$, $\delta = 4$

Experiments

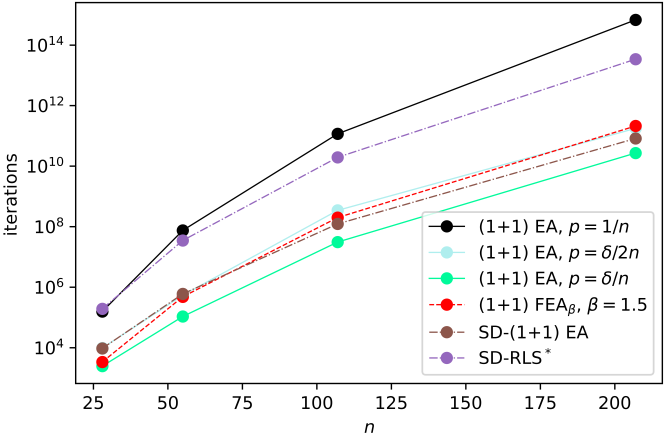


Figure: Optimization times of different algorithms on $\text{JUMP}_{k,\delta}$ with $k = 3\ln(n)$, $\delta = \frac{k}{2}$

Experiments

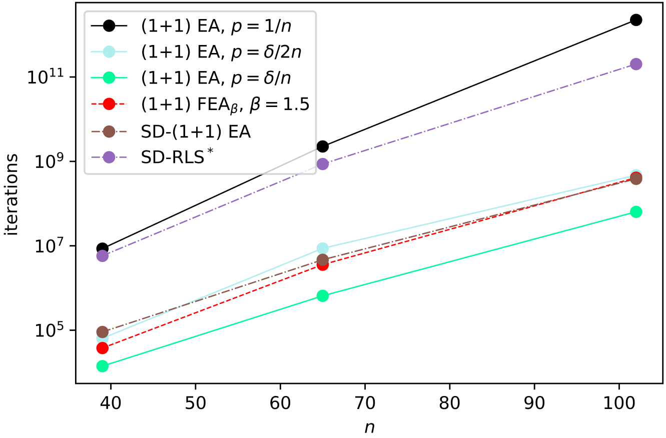


Figure: Optimization times of different algorithms on $\text{JUMP}_{k,\delta}$ with $k = 4n^{0.3}$, $\delta = \frac{k}{2}$.

Experiments

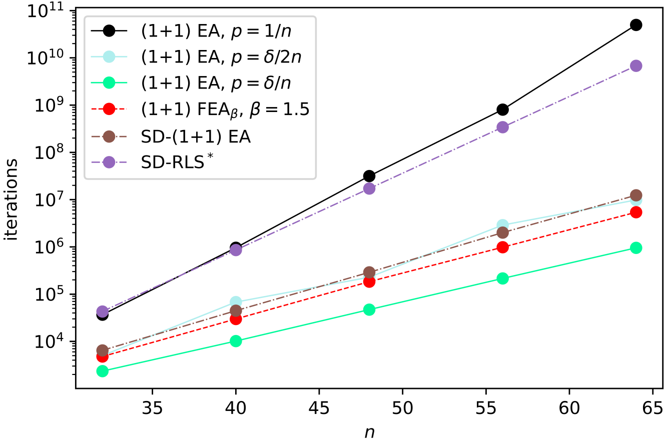
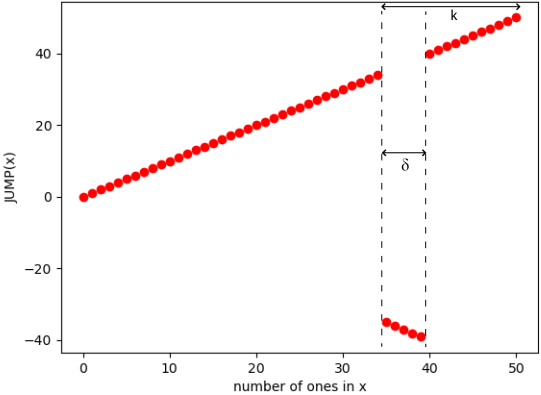


Figure: Optimization times of different algorithms on $\text{JUMP}_{k,\delta}$ with $k = \frac{n}{4}$, $\delta = \frac{n}{8}$.

Conclusion

Conclusion: The $\text{JUMP}_{k,\delta}$ function

$$\text{JUMP}_{k,\delta}(x) = \begin{cases} \|x\|_1 & \text{if } \|x\|_1 \in [0..n-k] \cup [n-k+\delta..n], \\ -\|x\|_1 & \text{otherwise.} \end{cases}$$



A more realistic version of the well-known JUMP_k function.

Conclusion: Performance of Algorithms

Algorithm	Jump_k	$\text{Jump}_{k,\delta}$ in the SR
$(1+1)$ EA with optimal MR	$\Theta((\frac{k}{n})^{-k}(\frac{n}{n-k})^{n-k})$ [DLMN17]	$(1+o(1))(\frac{en}{\delta})^{\delta}\binom{k}{\delta}^{-1}$
$(1+1)$ FEA_β	$O(C_{n/2}^{\beta}k^{\beta-0.5}(\frac{k}{n})^{-k}(\frac{n}{n-k})^{n-k})$ [DLMN17]	$O(C_{n/2}^{\beta}\delta^{\beta-0.5}(\frac{en}{\delta})^{\delta}\binom{k}{\delta}^{-1})$
SD-RLS*	$\binom{n}{k}(1+O(\frac{k^2}{n-2k}\ln(n)))$ [RW21]	$(1+o(1))[\ln(R)\sum_{i=1}^{\delta-1}\sum_0^i\binom{n}{j}+\binom{n}{\delta}\binom{k}{\delta}^{-1}]$
SD-$(1+1)$ EA	$O((\frac{en}{k})^k)$ [RW20]	Unclear

Conclusion: Performance of Algorithms

Algorithm	**Jump**$_k$	**Jump**$_{k,\delta}$ **in the SR**
$(1+1)$ EA with optimal MR	$\Theta((\frac{k}{n})^{-k}(\frac{n}{n-k})^{n-k})$	$(1+o(1))(\frac{en}{\delta})^{\delta}\binom{k}{\delta}^{-1}$
$(1+1)$ FEA$_\beta$	$k^{\beta-0.5}$	$\delta^{\beta-0.5}$
SD-RLS*	$\left(\frac{en}{k}\right)^k\binom{n}{k}^{-1}$	$\Omega(n^K), \forall K>0$
SD-$(1+1)$ EA	$\approx$	Unknown

References I

Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen, *Fast genetic algorithms*, Genetic and Evolutionary Computation Conference, GECCO 2017, ACM, 2017, pp. 777–784.

Amirhossein Rajabi and Carsten Witt, *Self-adjusting evolutionary algorithms for multimodal optimization*, Genetic and Evolutionary Computation Conference, GECCO 2020, ACM, 2020, pp. 1314–1322.

________, *Stagnation detection with randomized local search*, Evolutionary Computation in Combinatorial Optimization, EvoCOP 2021, Springer, 2021, pp. 152–168.

Ingo Wegener, *Theoretical aspects of evolutionary algorithms*, Automata, Languages and Programming, ICALP 2001, Springer, 2001, pp. 64–78.