Escaping Large Deceptive Basins of Attraction with Heavy-Tailed Mutation Operators

Tobias Friedrich, Francesco Quinzan, Markus Wagner

How to mutate? I mean: mutation rate, …?

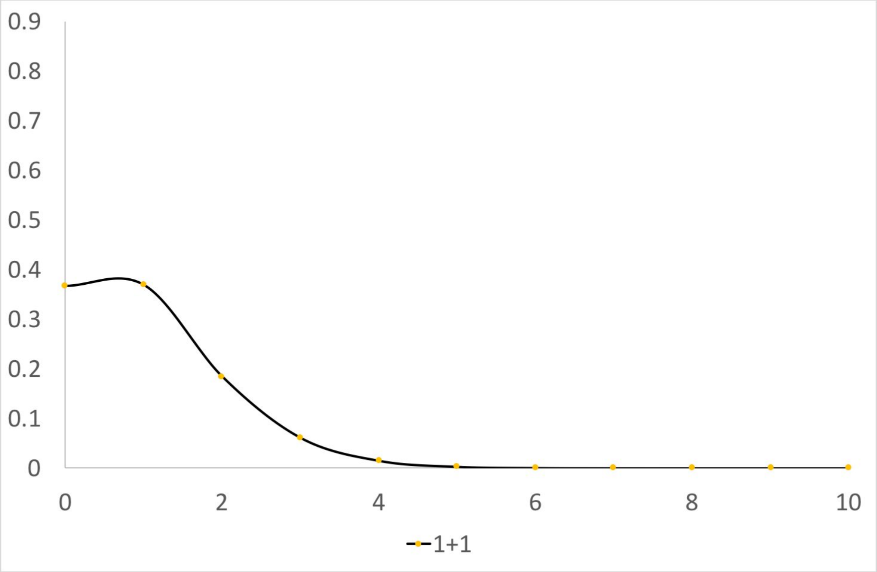
Many packages do this: if *n* is the length of a solution, then perform mutation with probability *1/n*.

Often found in theory: if *n* is the bitstring of length *n*, then flip each bit with *1/n*

How to mutate? I mean: mutation rate, …?

Many packages do this: if *n* is the length of a solution, then perform mutation with probability *1/n*.

Often found in theory: if *n* is the bitstring of length *n*, then flip each bit with *1/n*



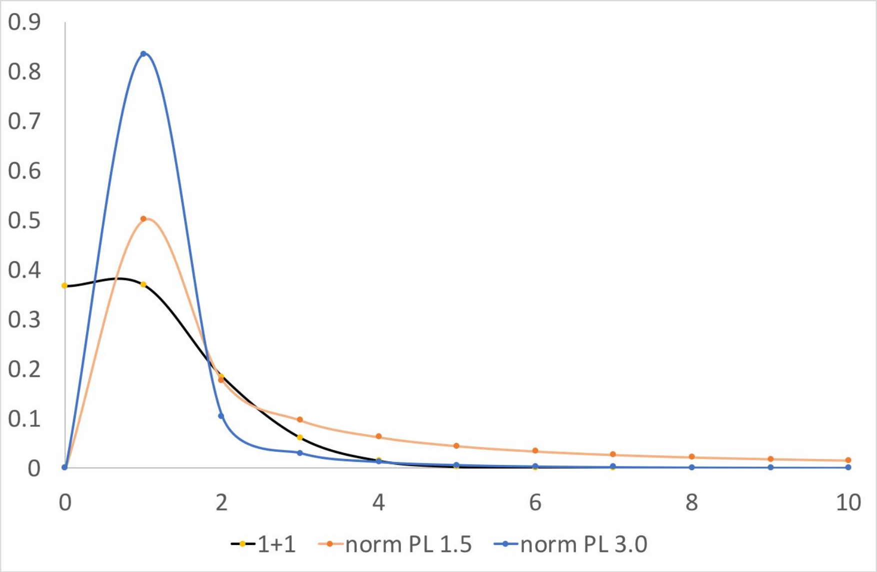
How to mutate? I mean: mutation rate, …?

Many packages do this: if *n* is the length of a solution, then perform mutation with probability *1/n*.

Often found in theory: if *n* is the bitstring of length *n*, then flip each bit with *1/n*

<u>GECCO’17</u>: theoretical study, where the number of flipped bits is drawn from a power law distribution

<u>Goal</u>: escape local optima



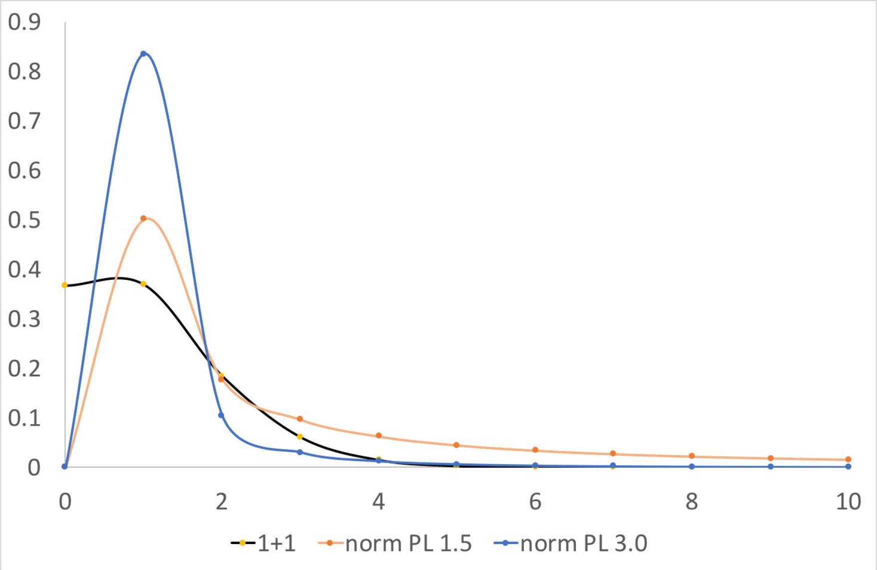
How to mutate? I mean: mutation rate, …?

Many packages do this: if *n* is the length of a solution, then perform mutation with probability *1/n*.

Often found in theory: if *n* is the bitstring of length *n*, then flip each bit with *1/n*

<u>GECCO'17</u>: theoretical study, where the number of flipped bits is drawn from a power law distribution

<u>Goal</u>: escape local optima



<u>This GECCO'18</u>: simpler operator, theory, experiments on minimum vertex cover + maximum cut

ps: there is already more at PPSN'18 :-) and at GECCO'18 tomorrow (GA3 session, Doerr/Wagner)

Preliminaries

Algorithm 1: General framework for the (1+1) EA

Choose initial solution $x \in \{0, 1\}^n$ u.a.r.;

Preliminaries

Algorithm 1: General framework for the (1+1) EA

Choose initial solution $x \in \{0, 1\}^n$ u.a.r.;
while *convergence criterion not met* **do**
    $y \leftarrow$ **Mutation**(x) for given mutation operator;
    if $f(y) \geq f(x)$ **then**
        $x \leftarrow y$;
return x;

Preliminaries

Doerr et al. GECCO'17
Intuitively: probability to perform a k-bit mutation is ~k^-β

Algorithm 1: General framework for the (1+1) EA

Choose initial solution $x \in \{0,1\}^n$ u.a.r.;
while *convergence criterion not met* **do**
    $y \leftarrow$ **Mutation**(x) for given mutation operator;
    if $f(y) \geq f(x)$ **then**
        $x \leftarrow y$;
return x;

Algorithm 2: The mutation operator $\text{fmut}_\beta(x)$

$y \leftarrow x$;
choose $k \in [1, \ldots, n/2]$ according to distribution $D^{\beta}_{n/2}$;
for $j = 1, \ldots, n$ **do**
    if ***random***$([0,1])n \leq k$ **then**
        $y[j] \leftarrow 1 - y[j]$;
return y;

Preliminaries

Doerr et al. GECCO'17
Intuitively: probability to perform a k-bit mutation is ~k^-β

Algorithm 1: General framework for the (1+1) EA

Choose initial solution $x \in \{0,1\}^n$ u.a.r.;
while *convergence criterion not met* **do**
  $y \leftarrow$ **Mutation**(x) for given mutation operator;
  if $f(y) \geq f(x)$ **then**
    $x \leftarrow y$;
return x;

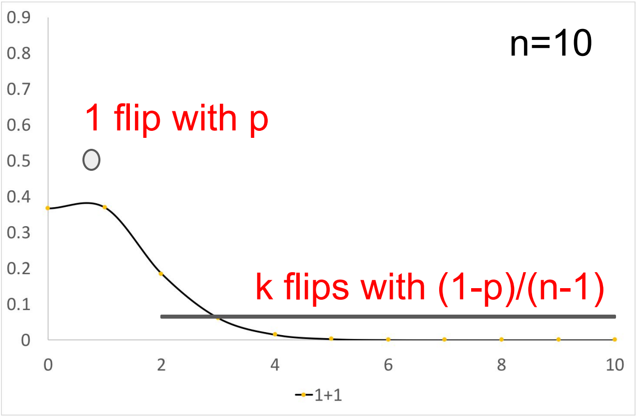
Algorithm 2: The mutation operator $\mathrm{fmut}_\beta(x)$

$y \leftarrow x$;
choose $k \in [1, \ldots, n/2]$ according to distribution $D^{\beta}_{n/2}$;
for $j = 1, \ldots, n$ **do**
  if ***random***$([0,1])n \leq k$ **then**
    $y[j] \leftarrow 1 - y[j]$;
return y;

This GECCO'18:

Algorithm 3: The mutation operator $\mathrm{cMut}_p(x)$

$y \leftarrow x, k \leftarrow 1$;
if ***random***$([0,1]) > p$ **then**
  choose $k \in \{2, \ldots, n\}$ u.a.r.;
flip k-bits of y chosen u.a.r.;
return y;



Theory

$$\text{OneMax}(x_1, \ldots, x_n) = |x|_1 = \sum_{j=1}^{n} x_j \Longrightarrow \mathcal{O}\left(\frac{n}{p} \log n\right)$$

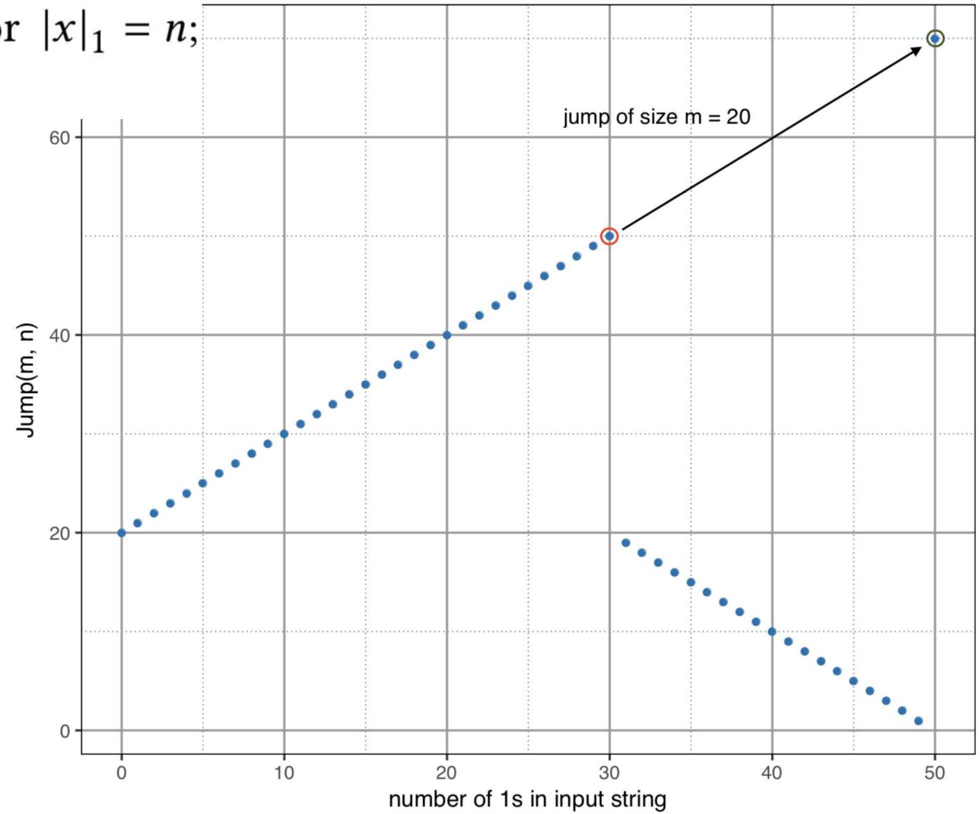
Lemma 3.1.

$$\text{OneMax}(x_1, \ldots, x_n) = |x|_1 = \textstyle\sum_{j=1}^{n} x_j \Longrightarrow$$

Lemma 3.1. $O\left(\frac{n}{p} \log n\right)$

Theory

$$\text{Jump}(m,n)(x) = \begin{cases} m + |x|_1 & \text{if } |x|_1 \leq n - m \text{ or } |x|_1 = n; \\ n - |x|_1 & \text{otherwise;} \end{cases}$$



n=50
m=20
→ 20-flip mutation needed!

Jump(m,n) - Doerr’s fmut (T_β) vs our cmut (T_p)

Lemma 3.6 if m is constant $$T_p(f) = \Theta(nT_\beta(f))$$

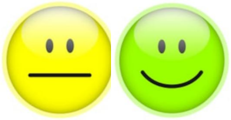
Jump(m,n) - Doerr's fmut (T_β) vs our cmut (T_p)

Lemma 3.6   if m is constant

$$T_p(f) = \Theta(nT_\beta(f)).$$

Lemma 3.7   if ...<=m<=n/2

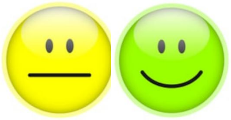
$$T_p(f) \leq \frac{c}{H_{n/2}^{(\beta)}} T_\beta(f)$$



Jump(m,n) - Doerr's fmut (T_β) vs our cmut (T_p)

Lemma 3.6 if m is constant $T_p(f) = \Theta(nT_\beta(f))$.

Lemma 3.7 if ...<=m<=n/2 $T_p(f) \leq \frac{c}{H_{n/2}^{(\beta)}} T_\beta(f)$



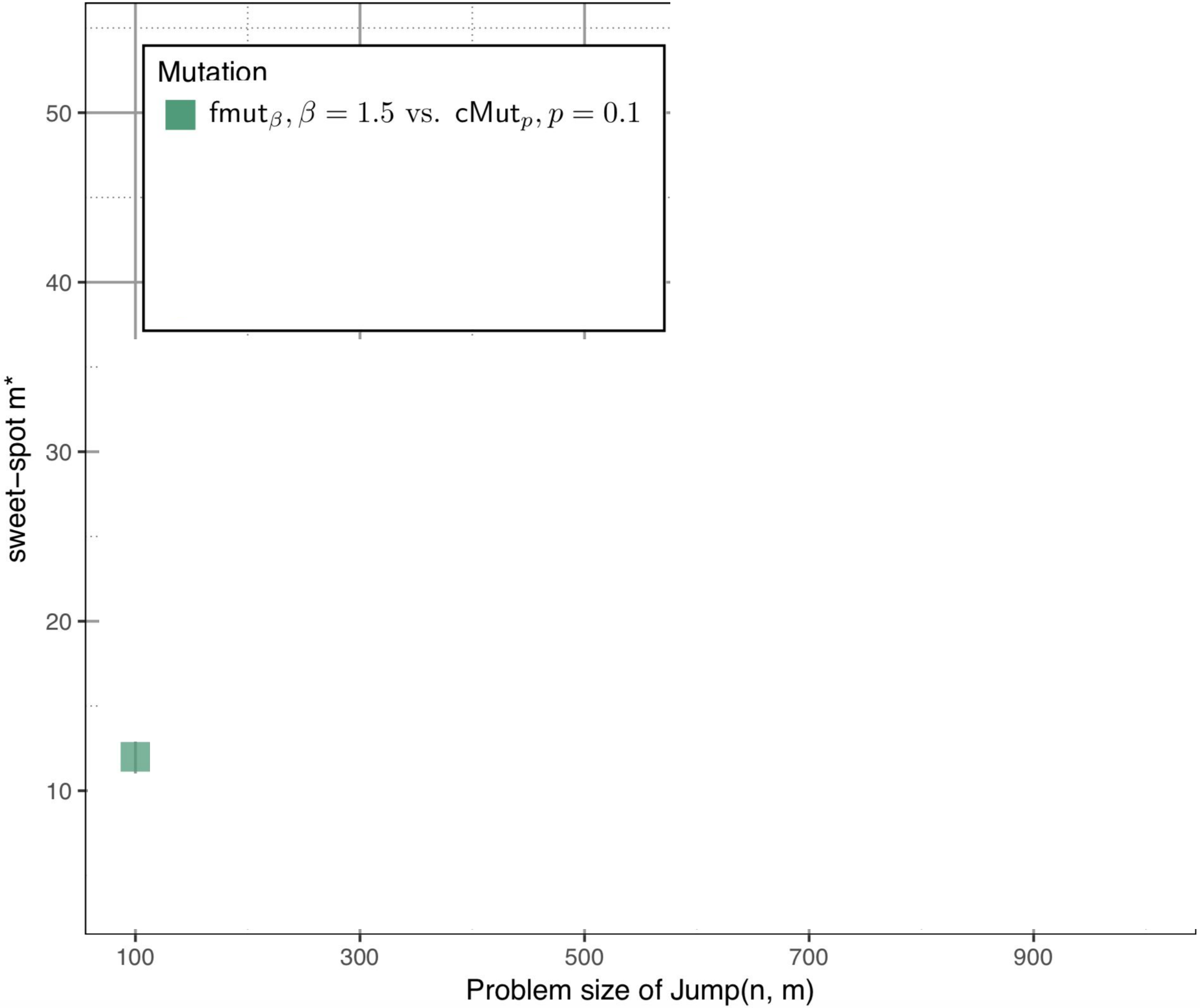
Lemma 3.8 if n-m is constant $T_\beta(f) = 2^{\Omega(n)}$ $T_p(f) = n^{\Theta(1)}$

⇒ There is a sweet spot m* s.t. cmut outperforms fmut on all Jump(n,m) with m>=m*

https://www.shutterstock.com/search/green+orange+face+smiley

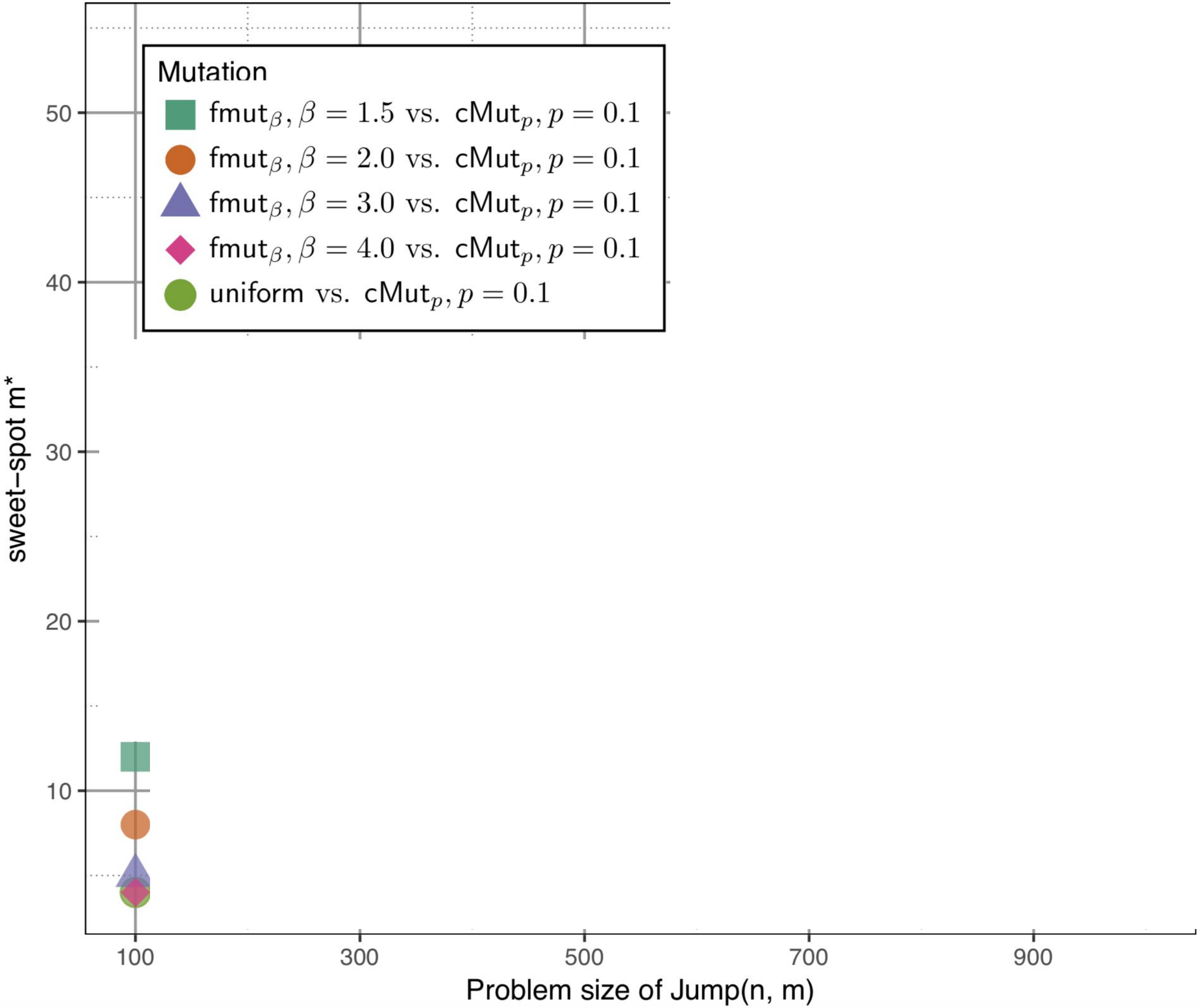
fmut vs our cmut: sweet spot m*

1. Solve Jump(n,m), various m (keep n fixed)
2. Determine from which m* on cmut is better than fmut



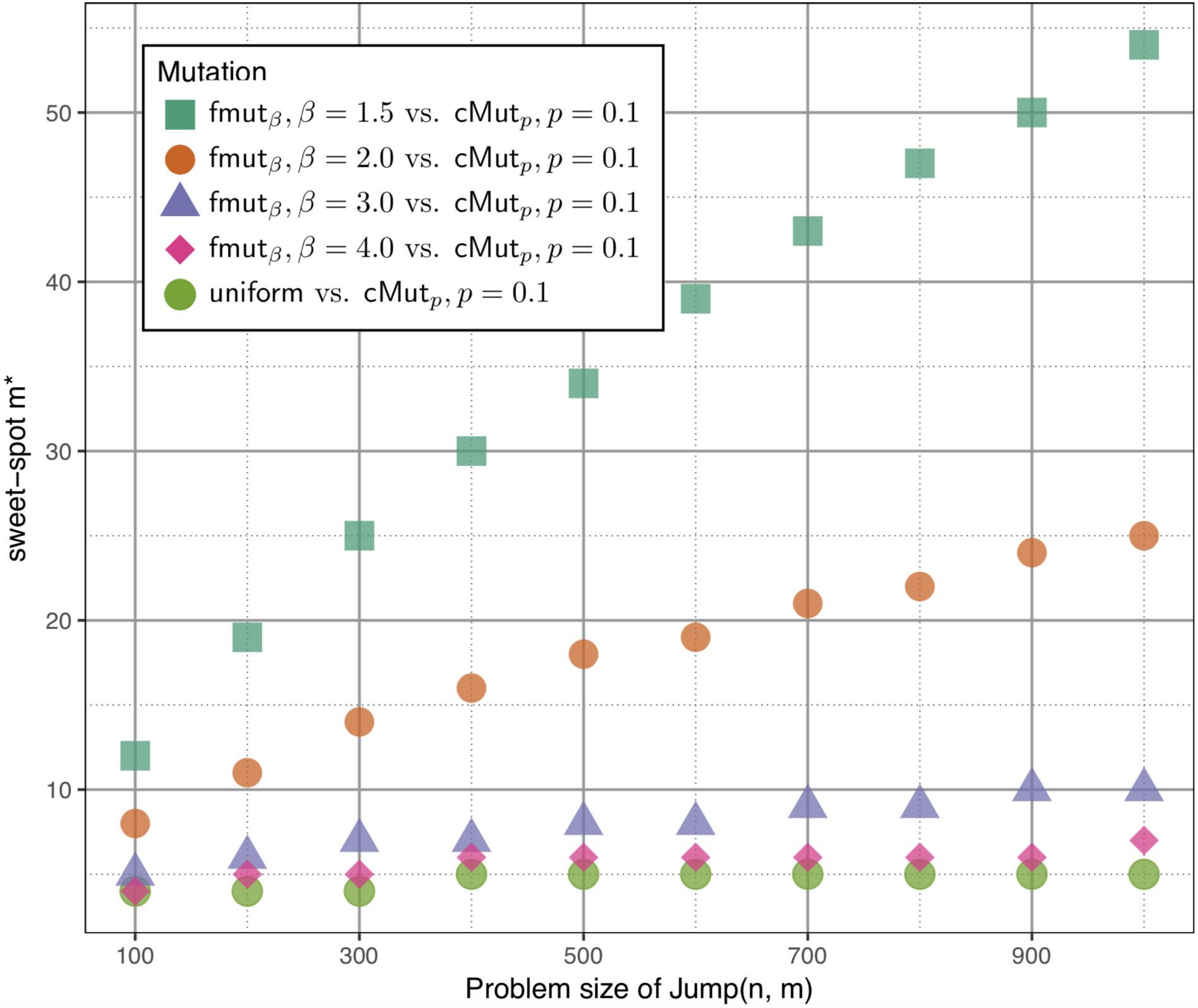
fmut vs our cmut: sweet spot m*

1. Solve Jump(n,m), various m (keep n fixed)
2. Determine from which m* on cmut is better than fmut



fmut vs our cmut: sweet spot m*

1. Solve Jump(n,m), various m (keep n fixed)
2. Determine from which m* on cmut is better than fmut



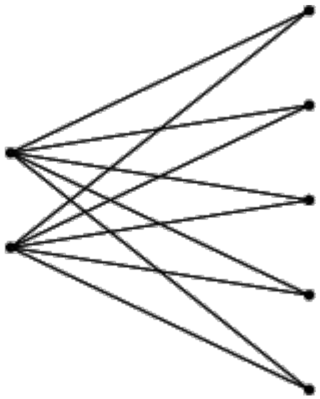
Theory, Minimum Vertex Cover

Given a graph G=(V,E) of order *n* find a minimal subset $U \subseteq V$ s.t. each edge in *E* is adjacent to at least one vertex.

For a given indexing on the vertices of *G*, each subset $U \subseteq V$ is represented as a pseudo-boolean array $(x_1,...,x_n)$ with $x_i = 1$ iff the *i*-th vertex is in *U*. Thus, in this context the problem size is the order of the graph.

We approach the MVC by minimizing the function $(u(x),|x|_1)$ in lexicographical order, with $u(x)$ the function that returns the number of uncovered edges. We restrict the analysis on complete bipartite graphs, defined as follows.

One example



https://archive.lib.msu.edu/crcmath/math/math/c/c475.htm

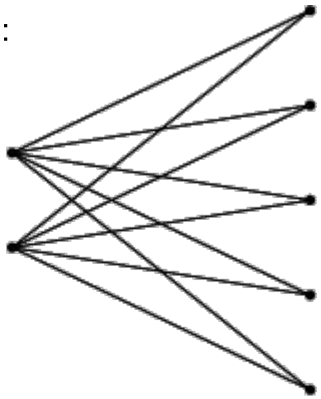
Theory, Minimum Vertex Cover

Given a graph G=(V,E) of order *n* find a minimal subset $U \subseteq V$ s.t. each edge in *E* is adjacent to at least one vertex.

For a given indexing on the vertices of *G*, each subset $U \subseteq V$ is represented as a pseudo-boolean array $(x_1,...,x_n)$ with $x_i = 1$ iff the *i*-th vertex is in *U*. Thus, in this context the problem size is the order of the graph.

We approach the MVC by minimizing the function $(u(x),|x|_1)$ in lexicographical order, with *u(x)* the function that returns the number of uncovered edges. We restrict the analysis on complete bipartite graphs, defined as follows.

One example:



Traditional (1+1)-EA with *1/n* performs poorly.

Theorem 4.2:

$$O\left(\frac{n}{p}\log n + \frac{n}{1-p}\right)$$

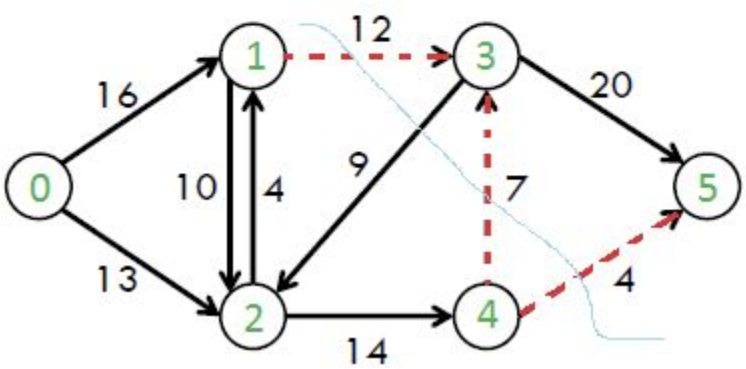
1. Phase: find a vertex cover in O(n log n)
2. Phase: kick out vertices in O(n/p log n)
3. Phase: done if optimal, otherwise flip with (1-p)/(n-1)

https://archive.lib.msu.edu/crcmath/math/math/c/c475.htm

Theory, Maximum Cut

Given a (directed) graph $G = (V,E)$: find a subset of vertices $U \subseteq V$ s.t. the sum of the weights edges leaving U is maximal.

One example:

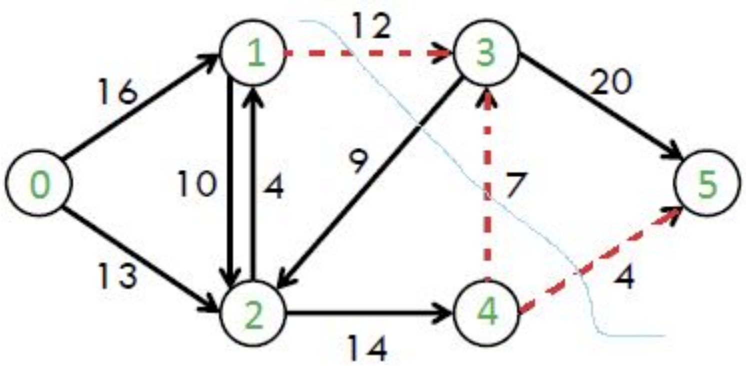


U here: {0,1,2,4}, cut: 12+7+4=23

Theory, Maximum Cut

Given a (directed) graph $G = (V,E)$: find a subset of vertices $U \subseteq V$ s.t. the sum of the weights edges leaving U is maximal.

One example:



U here: {0,1,2,4}, cut: 12+7+4=23

Previous work:

$$1/(3(1+\epsilon))\text{-approximation}$$

$$\mathcal{O}\left(\frac{1}{\epsilon} n^7 \log n\right)$$

Theorem 4.7:

$$(1/3-\epsilon/n)\text{-}approximation$$

$$\mathcal{O}\left(\frac{1}{\epsilon}\frac{n^3}{p}\log(n\Delta) + \frac{n}{1-p}\right)$$

max out degree

https://www.geeksforgeeks.org/wp-content/uploads/minCut.png

Experiments - Evolving the distribution

Automated algorithm configuration using irace (irated racing of configurations).

Result when evolving for the family of Jump functions with n=10, m=1..5:

```
# ACDT: Elite configurations (first number is the configuration ID):
```

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
5599	0.70	0.03	0.03	0.02	0.02	0.02	0.04	0.04	0.02	0.06
8176	0.69	0.07	0.04	0.02	0.01	0.01	0.02	0.07	0.02	0.06
6578	0.70	0.02	0.02	0.02	0.04	0.04	0.06	0.01	0.02	0.07
8991	0.71	0.04	0.03	0.01	0.06	0.04	0.02	0.02	0.01	0.05
9143	0.75	0.02	0.00	0.01	0.02	0.00	0.04	0.04	0.03	0.08

n=10

Looks like cmut, with p=0.70 and the rest is “evenly” distributed.

Experiments - Evolving the distribution

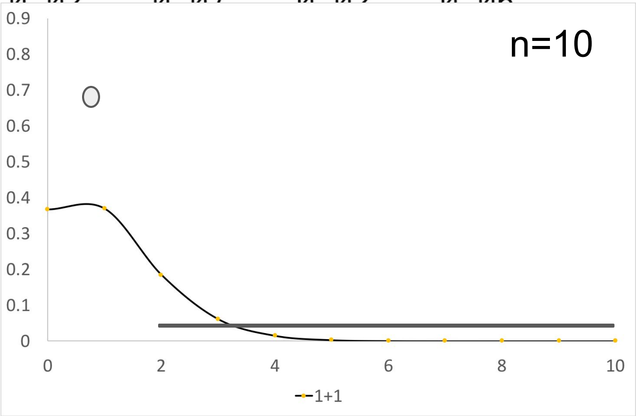
Automated algorithm configuration using irace (irated racing of configurations).

Result when evolving for the <u>family</u> of Jump functions with n=10, m=1..5:

ACDT: Elite configurations (first number is the configuration ID):

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
5599	0.70	0.03	0.03	0.02	0.02	0.02	0.04	0.04	0.02	0.06
8176	0.69	0.07	0.04	0.02	0.01	0.01	0.02	0.07	0.02	0.06
6578	0.70	0.02	0.02	0.02	0.04	0.04	[illegible]	[illegible]	[illegible]	[illegible]
8991	0.71	0.04	0.03	0.01	0.06	0.04	[illegible]	[illegible]	[illegible]	[illegible]
9143	0.75	0.02	0.00	0.01	0.02	0.00	[illegible]	[illegible]	[illegible]	[illegible]

Looks like cmut, with p=0.70 and the rest is “evenly”



Experiments - MaxCut, complete bipartite graphs

Weights:
going from left to right: 1.00
going from right to left: 1.01
n=100 (50 left, 50 right) →
optimum is 2525

mutation	1000 steps	10000 steps
uniform	2263.7	2500.0
$\text{fmut}_{1.5}$	2503.4	2513.8
$\text{fmut}_{2.0}$	2513.1	2514.6
$\text{fmut}_{3.0}$	2514.1	2514.6
$\text{fmut}_{4.0}$	2166.7	2514.1
$\text{cMut}_{0.1}$	2511.5	2525.0
$\text{cMut}_{0.5}$	2521.5	2525.0
$\text{cMut}_{0.9}$	2521.5	2525.0

Experiments - MaxCut, complete bipartite graphs

Weights:
going from left to right: 1.00
going from right to left: 1.01
n=100 (50 left, 50 right) → optimum is 2525

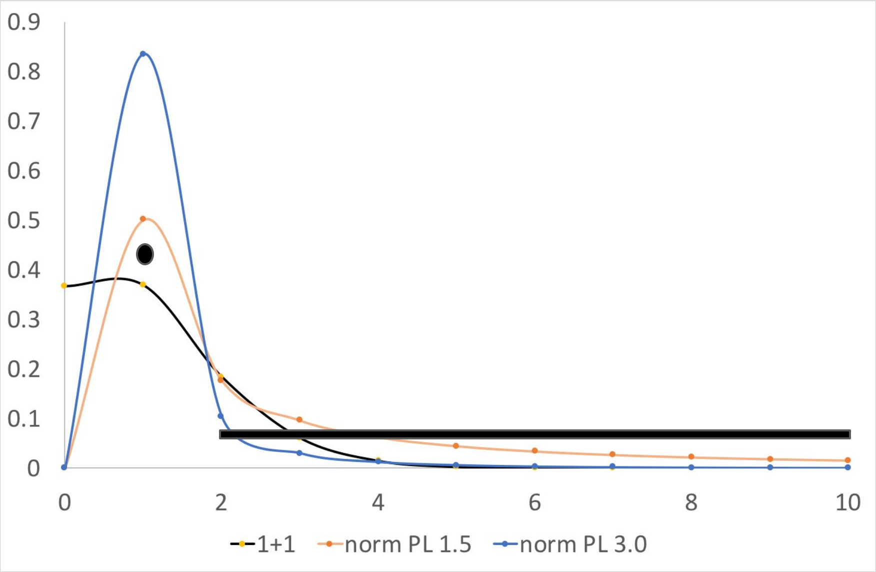
mutation	1000 steps	10000 steps
uniform	2263.7	2500.0
$fmut_{1.5}$	2503.4	2513.8
$fmut_{2.0}$	2513.1	2514.6
$fmut_{3.0}$	2514.1	2514.6
$fmut_{4.0}$	2166.7	2514.1
$cMut_{0.1}$	2511.5	2525.0
$cMut_{0.5}$	2521.5	2525.0
$cMut_{0.9}$	2521.5	2525.0

Sparse graphs with densities 0.5 and 0.1

mutation	1000 steps	10000 steps
uniform	1141.7	1247.4
$fmut_{1.5}$	1255.8	1261.2
$fmut_{2.0}$	1257.1	1258.3
$fmut_{3.0}$	1259.3	1259.7
$fmut_{4.0}$	1261.2	1261.3
$cMut_{0.1}$	1042.4	1269.9
$cMut_{0.5}$	1267.6	1273.2
$cMut_{0.9}$	1269.4	1271.2

mutation	1000 steps	10000 steps
uniform	226.4	255.2
$fmut_{1.5}$	251.9	255.9
$fmut_{2.0}$	249.9	253.0
$fmut_{3.0}$	249.7	231.3
$fmut_{4.0}$	251.5	252.3
$cMut_{0.1}$	202.4	256.3
$cMut_{0.5}$	246.4	255.8
$cMut_{0.9}$	251.8	255.3

Summary: How to mutate?



<u>This GECCO’18 paper</u>:
simpler operator, theory, experiments on minimum vertex cover + maximum cut

ps: there is already more at PPSN’18 :-) and at GECCO’18 tomorrow [GA3 session, Doerr/Wagner: super simple scheme for near-optimal mutation rates]