合肥學院
HEFEI UNIVERSITY

Metaheuristics for Smart Manufacturing

4. Stochastic Hill Climbing

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

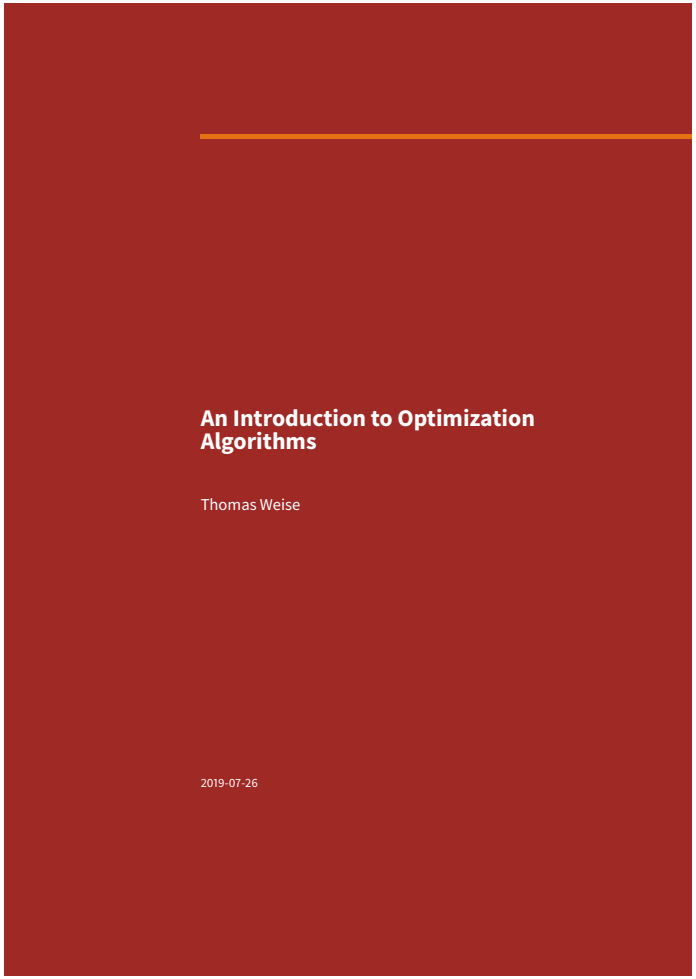
The slides are available at http://iao.hfuu.edu.cn/155, the book at http://thomasweise.github.io/aitoa, and the source code at http://www.github.com/thomasWeise/aitoa-code

course book

course material

An Introduction to Optimization Algorithms

The contents of this course are available as free electronic book *"An Introduction to Optimization Algorithms"* [1] at http://thomasweise.github.io/aitoa in pdf, html, azw3, and epub format, created with our bookbuildeR tool chain.



Section Outline

Introduction

- Our first algorithm, random sampling, was not very efficient.

Introduction

- Our first algorithm, random sampling, was not very efficient.
- It does not make any use of the information it "sees" during the optimization process.

Introduction

- Our first algorithm, random sampling, was not very efficient.
- It does not make any use of the information it "sees" during the optimization process.
- A search step consists of creating an entirely new, entirely random candidate solution.

Introduction

- Our first algorithm, random sampling, was not very efficient.
- It does not make any use of the information it "sees" during the optimization process.
- A search step consists of creating an entirely new, entirely random candidate solution.
- Every search step is thus independent of all prior steps.

Introduction

- Our first algorithm, random sampling, was not very efficient.
- It does not make any use of the information it "sees" during the optimization process.
- A search step consists of creating an entirely new, entirely random candidate solution.
- Every search step is thus independent of all prior steps.
- So how we can make use of the information we have seen during the search?

Introduction

- Our first algorithm, random sampling, was not very efficient.
- It does not make any use of the information it "sees" during the optimization process.
- A search step consists of creating an entirely new, entirely random candidate solution.
- Every search step is thus independent of all prior steps.
- So how we can make use of the information we have seen during the search?
- Instead of generating a completely random new candidate solution in each step...

Introduction

- Our first algorithm, random sampling, was not very efficient.
- It does not make any use of the information it "sees" during the optimization process.
- A search step consists of creating an entirely new, entirely random candidate solution.
- Every search step is thus independent of all prior steps.
- So how we can make use of the information we have seen during the search?
- Instead of generating a completely random new candidate solution in each step...
- ...why can't we try to iteratively improve the best solution we have seen so far?

Introduction

- Instead of generating a completely random new candidate solution in each step...
- ...why can't we try to iteratively improve the best solution we have seen so far?

Section Outline

Stoachstic Hill Climbing

- This is the concept of Local Search [2–5] and its simplest realization is Stochastic Hill Climbing [2].

Stoachstic Hill Climbing

- This is the concept of Local Search [2–5] and its simplest realization is Stochastic Hill Climbing [2].
- Simple Concept:
  1. create random initial solution

Stoachstic Hill Climbing

- This is the concept of Local Search [2–5] and its simplest realization is Stochastic Hill Climbing [2].
- Simple Concept:
  1. create random initial solution
  2. make a modified copy of best-so-far solution

Stoachstic Hill Climbing

- This is the concept of Local Search [2–5] and its simplest realization is Stochastic Hill Climbing [2].
- Simple Concept:
  1. create random initial solution
  2. make a modified copy of best-so-far solution
  3. if it is better, it becomes the new best-so-far solution (if it is not better, discard it).

Stoachstic Hill Climbing

- This is the concept of Local Search [2–5] and its simplest realization is Stochastic Hill Climbing [2].
- Simple Concept:
  1. create random initial solution
  2. make a modified copy of best-so-far solution
  3. if it is better, it becomes the new best-so-far solution (if it is not better, discard it).
  4. go back to 2 (until the time is up)

Causality

- Local searches like hill climbers exploit a property of many optimization problems called causality [6–9].

Causality

- Local searches like hill climbers exploit a property of many optimization problems called causality [6–9].
- Causality means that small changes in the features of an object (or candidate solution) also lead to small changes in its behavior (or objective value)

Causality

- Local searches like hill climbers exploit a property of many optimization problems called causality [6–9].
- Causality means that small changes in the features of an object (or candidate solution) also lead to small changes in its behavior (or objective value)
- The idea is that if we have a good candidate solution, then there may exist similar solutions which are better.

Causality

- Local searches like hill climbers exploit a property of many optimization problems called causality [6–9].
- Causality means that small changes in the features of an object (or candidate solution) also lead to small changes in its behavior (or objective value)
- The idea is that if we have a good candidate solution, then there may exist similar solutions which are better.
- We hope to find one of them and then continue trying to do the same from there.

Create Modified Copy of Existing Solution

- But how can we create a modified copy of an existing solution?

Create Modified Copy of Existing Solution

- But how can we create a modified copy of an existing solution?
- We cannot change the number of times any job appears in a string.

Create Modified Copy of Existing Solution

- But how can we create a modified copy of an existing solution?
- We cannot change the number of times any job appears in a string.
- Simple idea:
  - We randomly pick two indices i and j in the string and swap the job IDs at them.

Create Modified Copy of Existing Solution

- But how can we create a modified copy of an existing solution?
- We cannot change the number of times any job appears in a string.
- Simple idea:
  - We randomly pick two indices i and j in the string and swap the job IDs at them.
  - To make sure that the result is different, we can first check if the job IDs are different and if not, pick two new indices i and j.

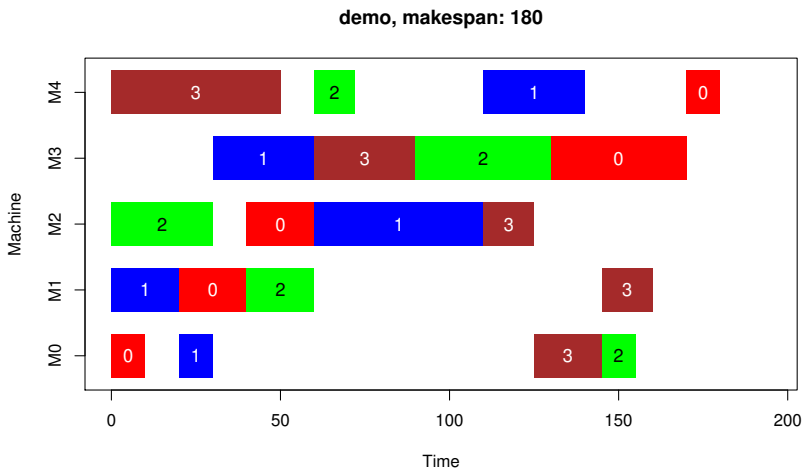
Create Modified Copy of Existing Solution

Example on our demo Problem Instance

```
(2, 0, 1, 0, 1, 1, 2, 3, 2, 3,
 2, 0, 0, 1, 3, 3, 2, 3, 1, 0)
```

Example on our demo Problem Instance

```
(2, 0, 1, 0, 1, 1, 2, 3, 2, 3,
 2, 0, 0, 1, 3, 3, 2, 3, 1, 0)
```



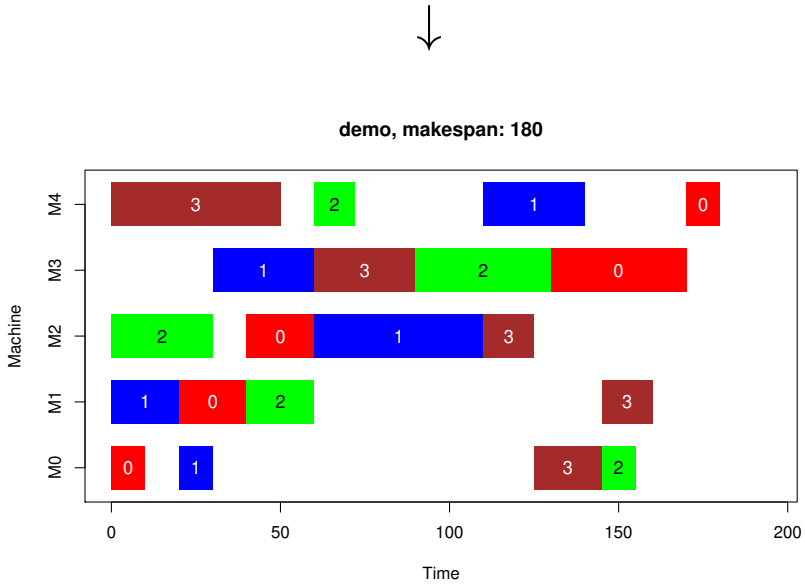
Create Modified Copy of Existing Solution

Example on our demo Problem Instance

(2, 0, 1, 0, 1, 1, 2, 3, 2, 3,
2, 0, 0, 1, 3, 3, 2, 3, 1, 0)

$\Longrightarrow$

(2, 0, 1, 0, 1, 1, 2, 3, 2, 3,
2, 0, 3, 1, 3, 0, 2, 3, 1, 0)



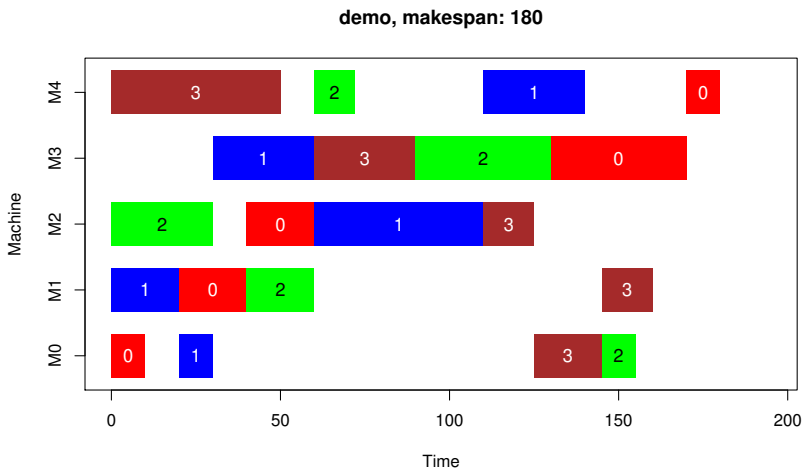
Create Modified Copy of Existing Solution

Example on our demo Problem Instance

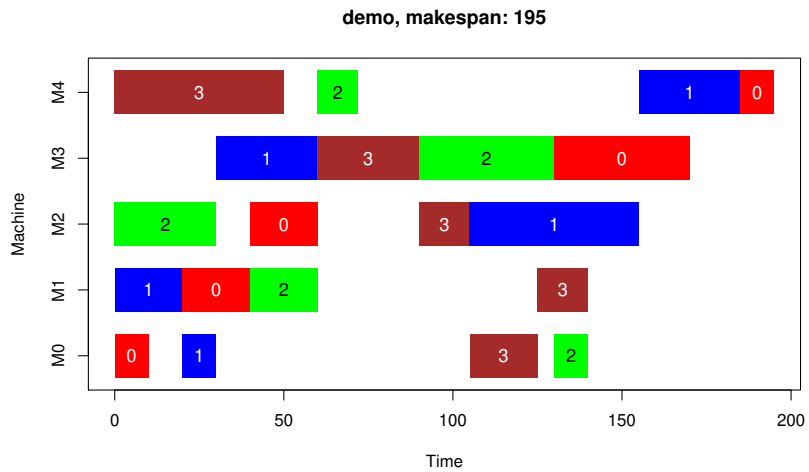
(2, 0, 1, 0, 1, 1, 2, 3, 2, 3,
2, 0, 0, 1, 3, 3, 2, 3, 1, 0)

$\Longrightarrow$

(2, 0, 1, 0, 1, 1, 2, 3, 2, 3,
2, 0, 3, 1, 3, 0, 2, 3, 1, 0)



$\Longrightarrow$



Create Modified Copy of Existing Solution

- But how can we create a modified copy of an existing solution?
- We cannot change the number of times any job appears in a string.
- Simple idea:
  - We randomly pick two indices i and j in the string and swap the job IDs at them.
  - To make sure that the result is different, we can first check if the job IDs are different and if not, pick two new indices i and j.
- Many modifications will lead to worse results, but some can be improvements.

Interface for Creating Modified Copy of Solution

Listing: An Java interface for generating a modified copy of a solution.

```
public interface IUnarySearchOperator<X> {

  public abstract void apply(X x, X dest, Random random);

}
```

Create Modified Copy of Existing Solution

Listing: Swap sub-jobs of two different jobs.

```
public class JSSPUnaryOperator1Swap
    implements IUnarySearchOperator<int[]> {

  public void apply(int[] x, int[] dest, Random random) {
    System.arraycopy(x, 0, dest, 0, x.length);

    int i = random.nextInt(dest.length);
    int job_i = dest[i];

    for (;;) {
      int j = random.nextInt(dest.length);
      int job_j = dest[j];
      if (job_i != job_j) {
        dest[i] = job_j;
        dest[j] = job_i;
        return;
      }
    }
  }
}
```

So what do we get?

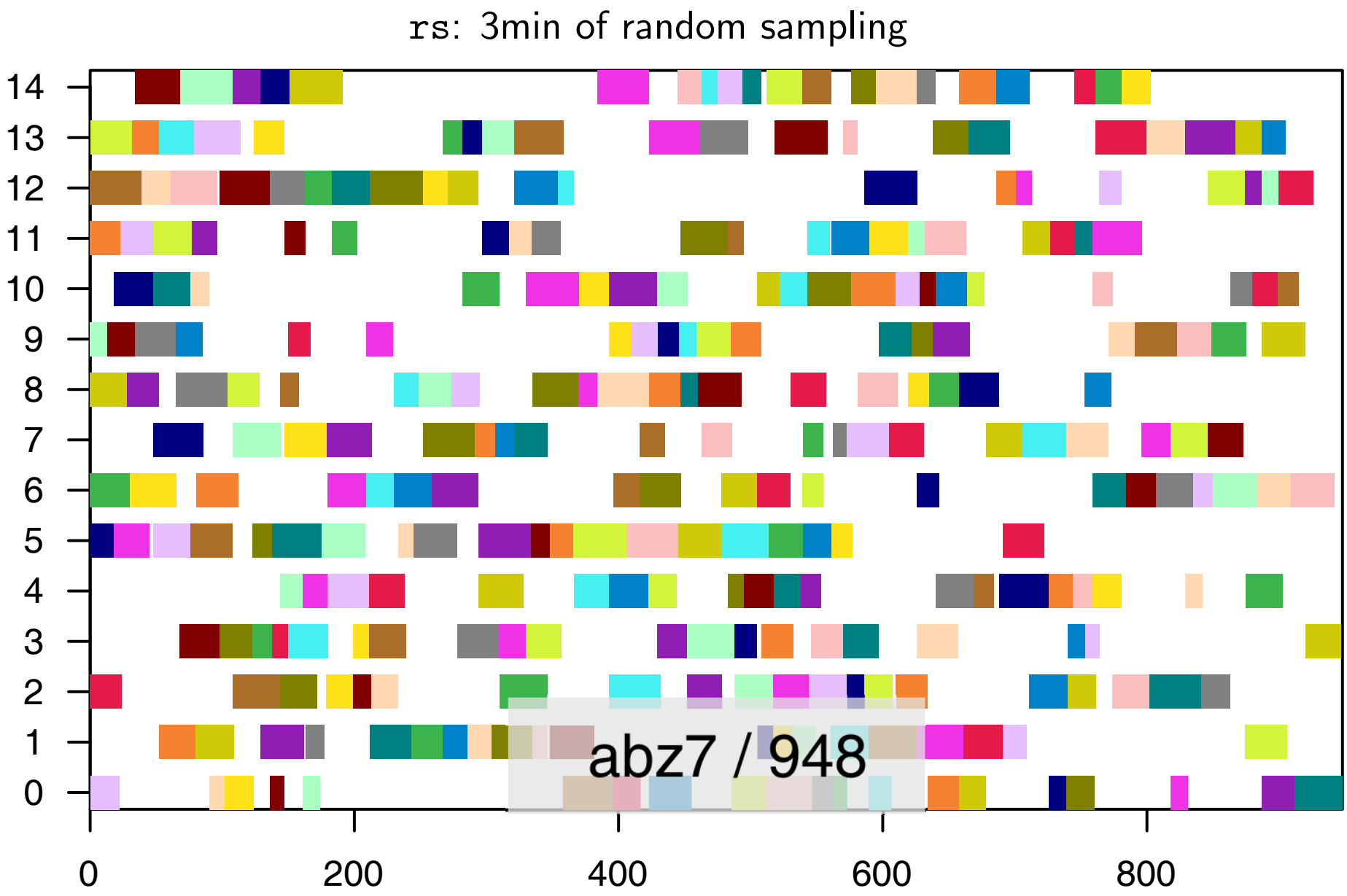
- I execute the program 101 times for each of the datasets `abz7`, `la24`, `swv15`, and `yn4`

So what do we get?

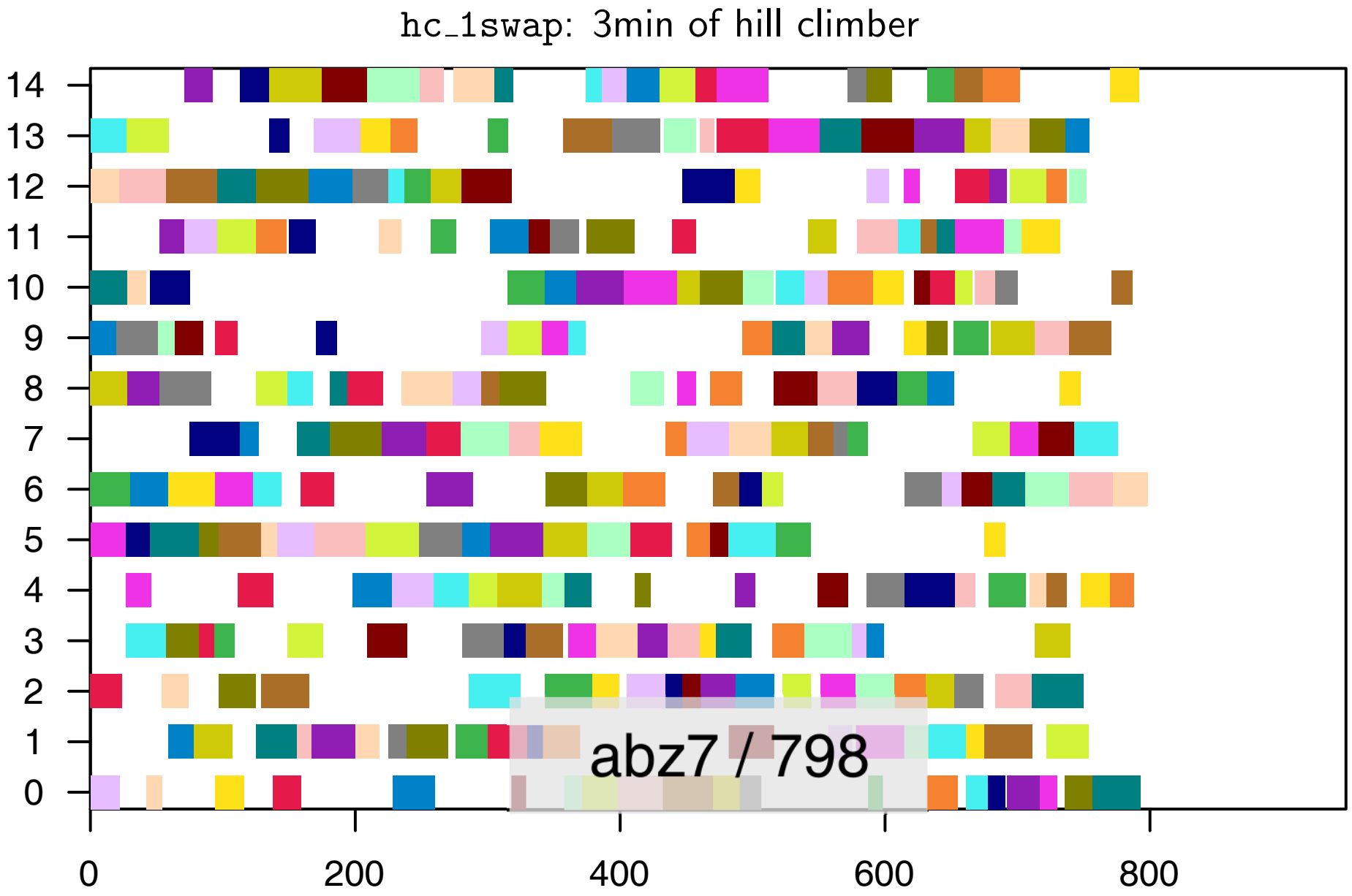
- I execute the program 101 times for each of the datasets `abz7`, `la24`, `swv15`, and `yn4`

		makespan				**last improvement**	
$\mathcal{I}$	algo	best	mean	med	sd	med(t)	med(FEs)
`abz7`	`rs`	895	945	948	**12**	77s	8'246'019
	`hc_1swap`	**717**	**800**	**798**	28	0s	16'978
`la24`	`rs`	1154	1206	1207	**15**	81s	17'287'329
	`hc_1swap`	**999**	**1095**	**1086**	56	0s	6612
`swv15`	`rs`	4988	5165	5174	**49**	85s	5'525'082
	`hc_1swap`	**3837**	**4108**	**4108**	137	1s	104'598
`yn4`	`rs`	1459	1496	1498	**15**	83s	6'549'694
	`hc_1swap`	**1109**	**1222**	**1220**	48	0s	31'789

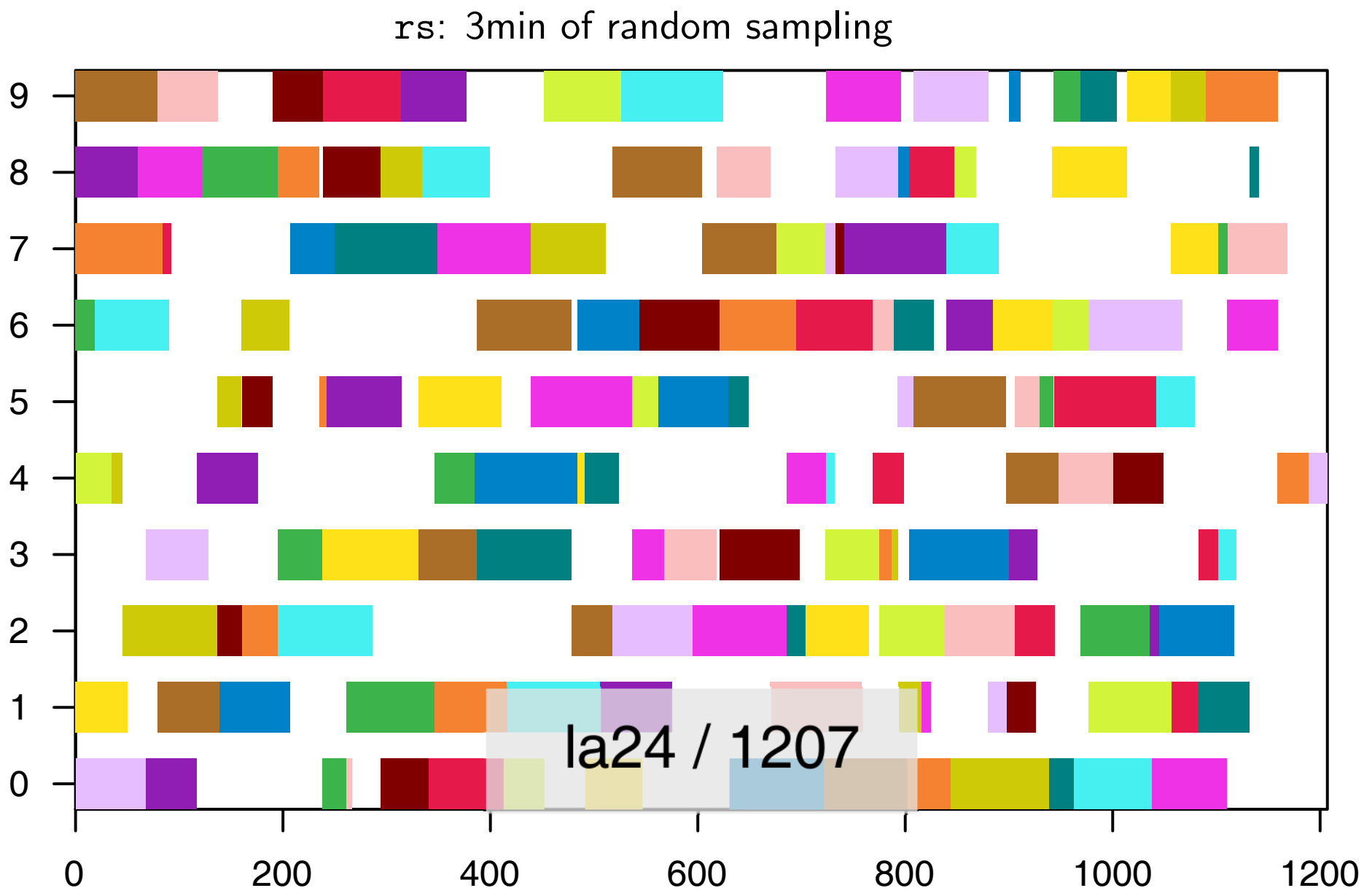
So what do we get?



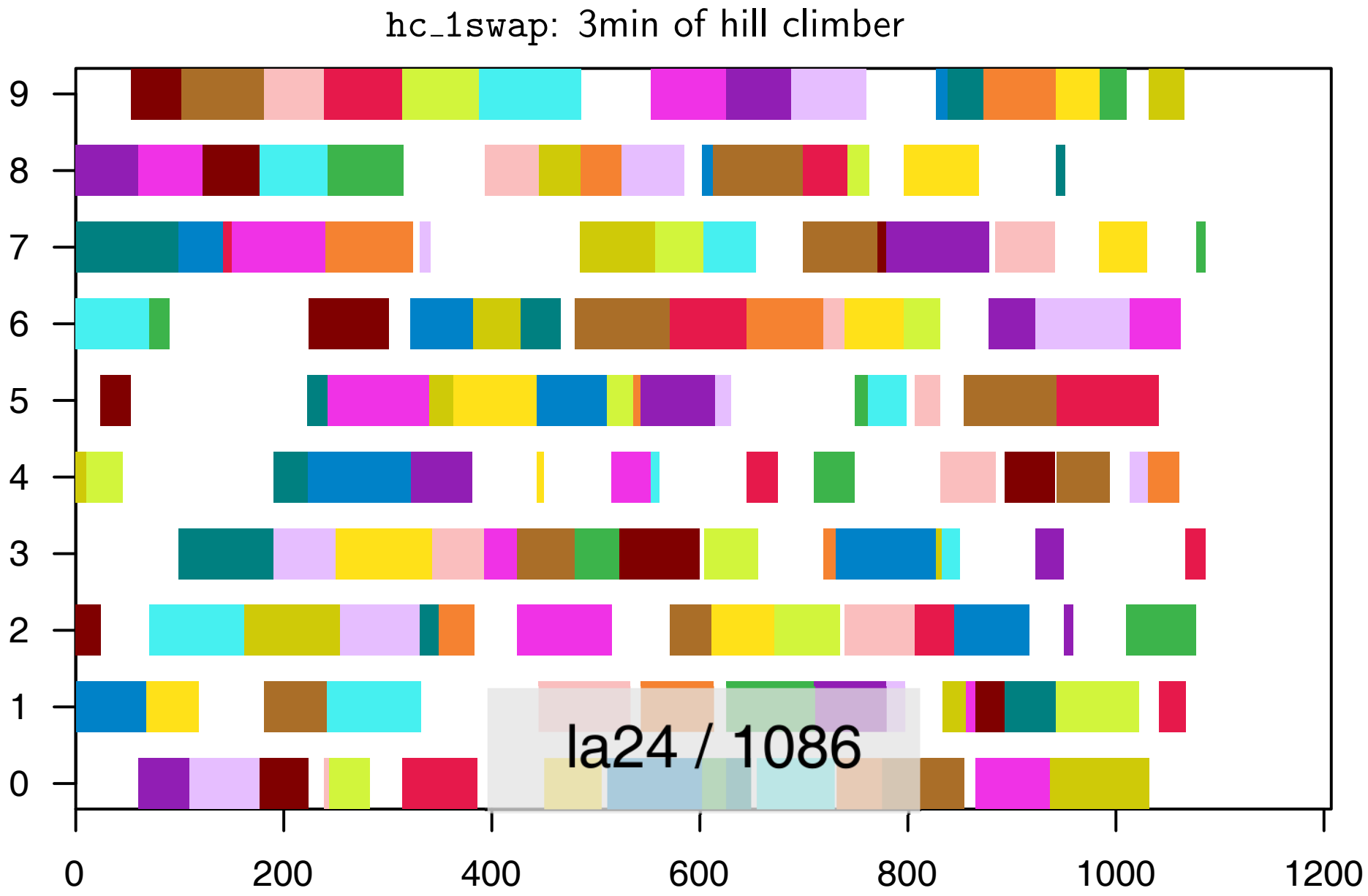
So what do we get?



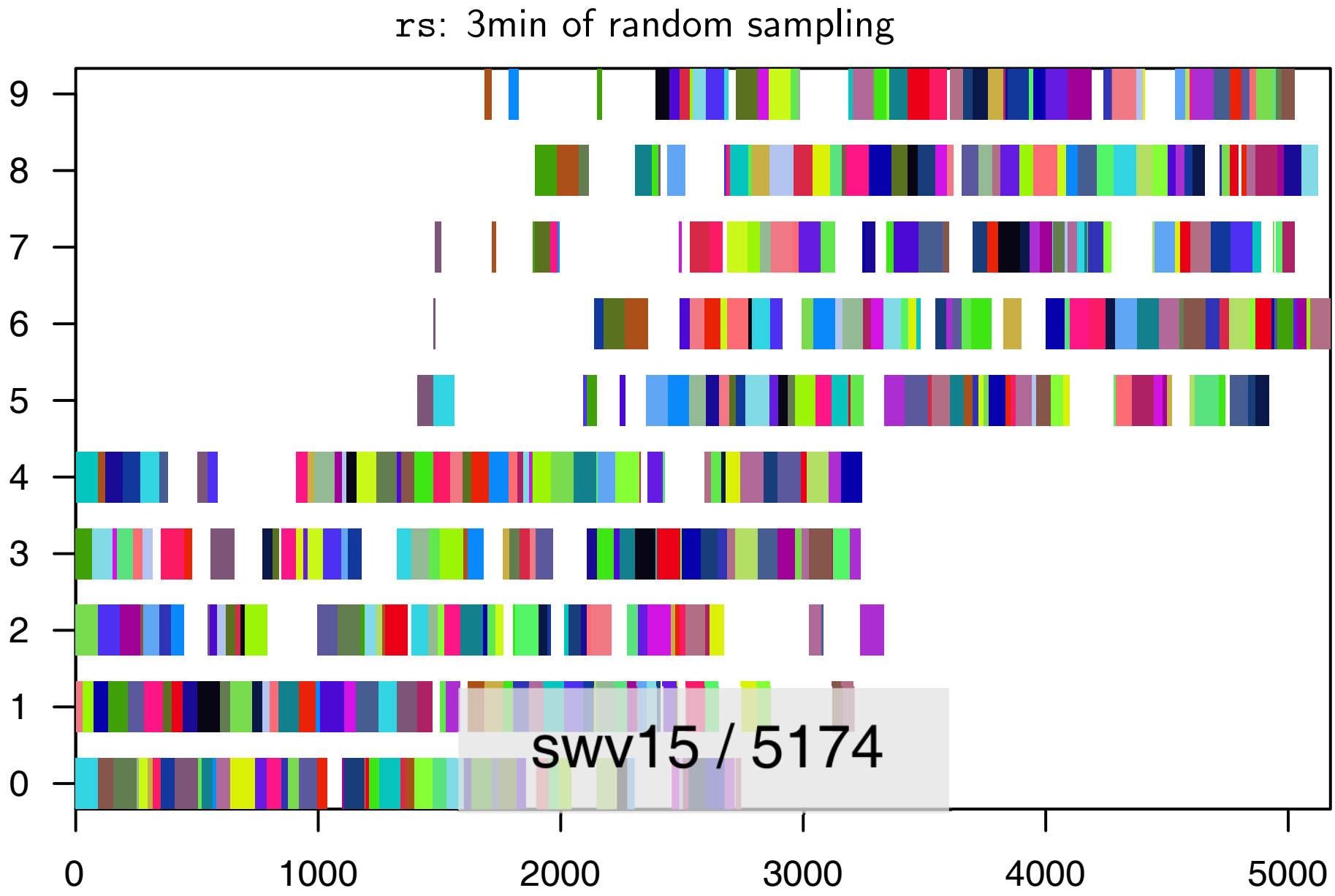
So what do we get?



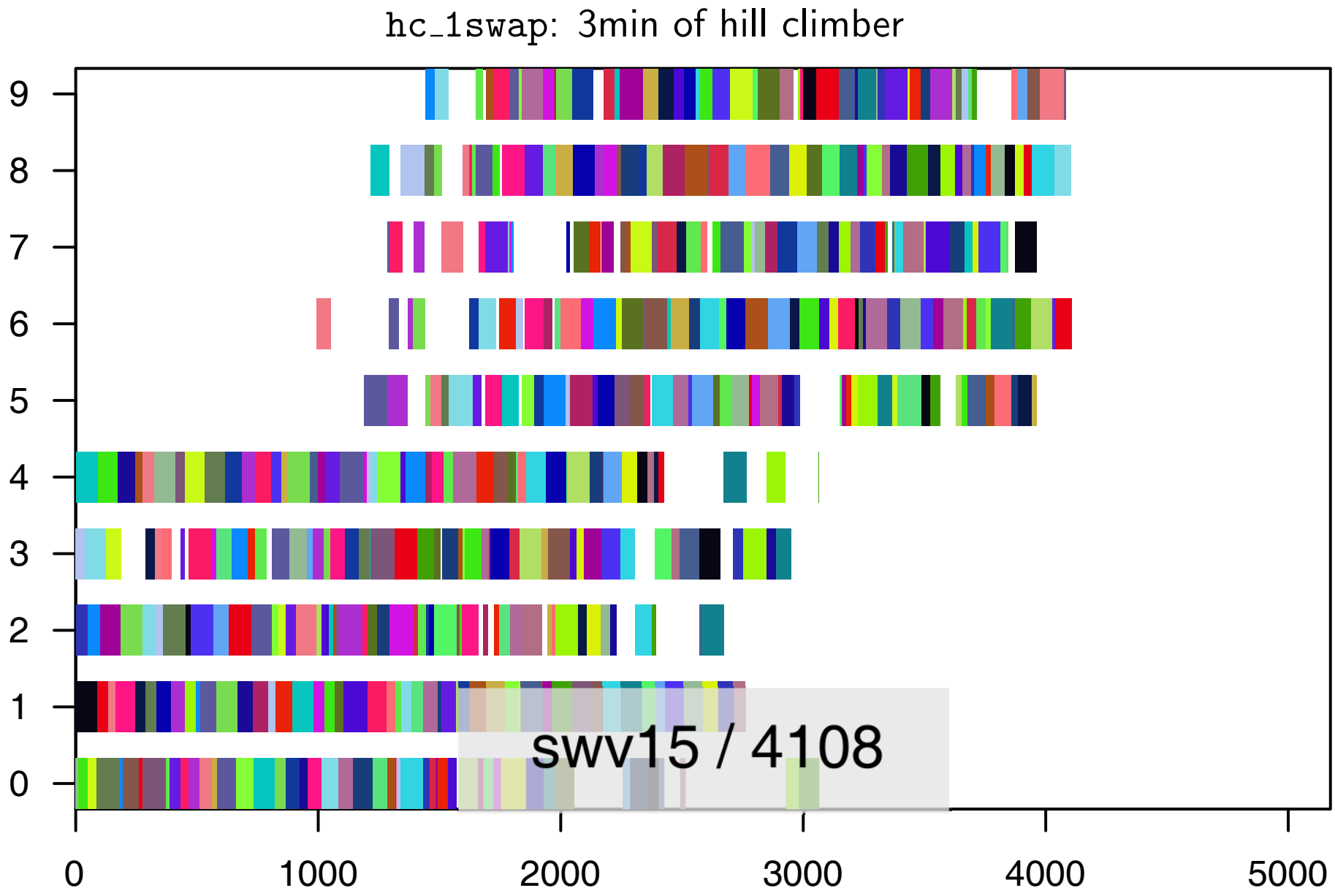
So what do we get?



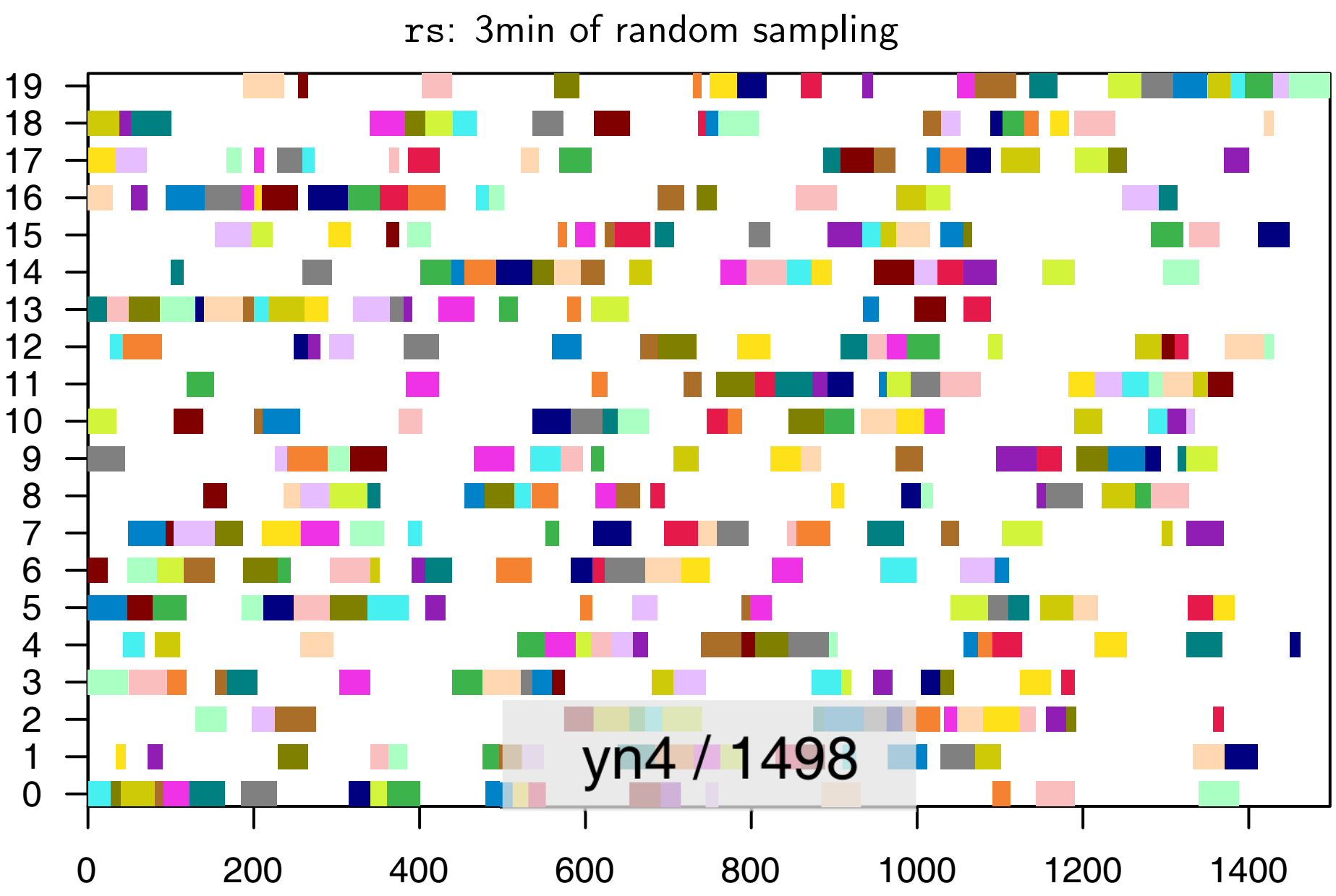
So what do we get?



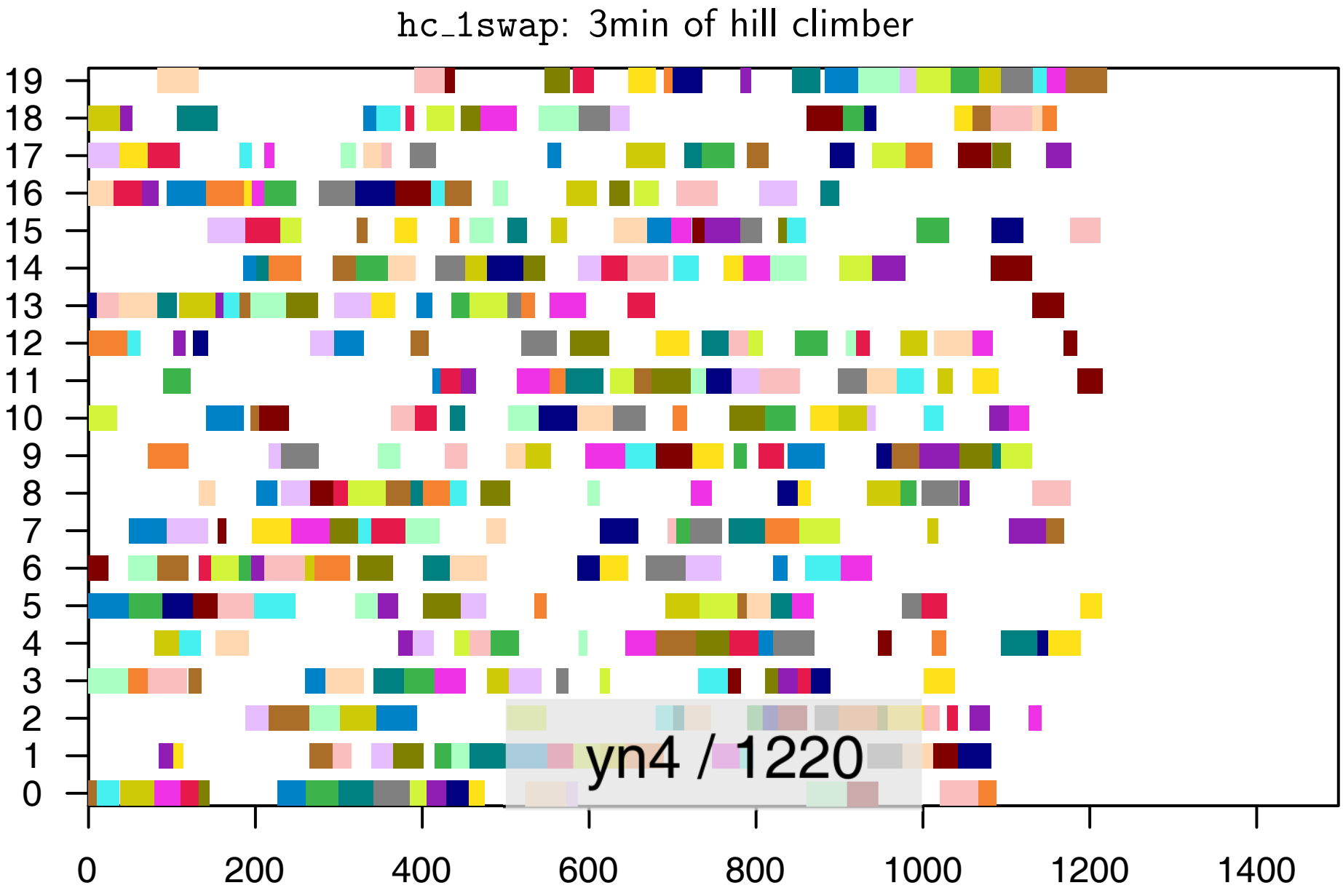
So what do we get?



So what do we get?



So what do we get?

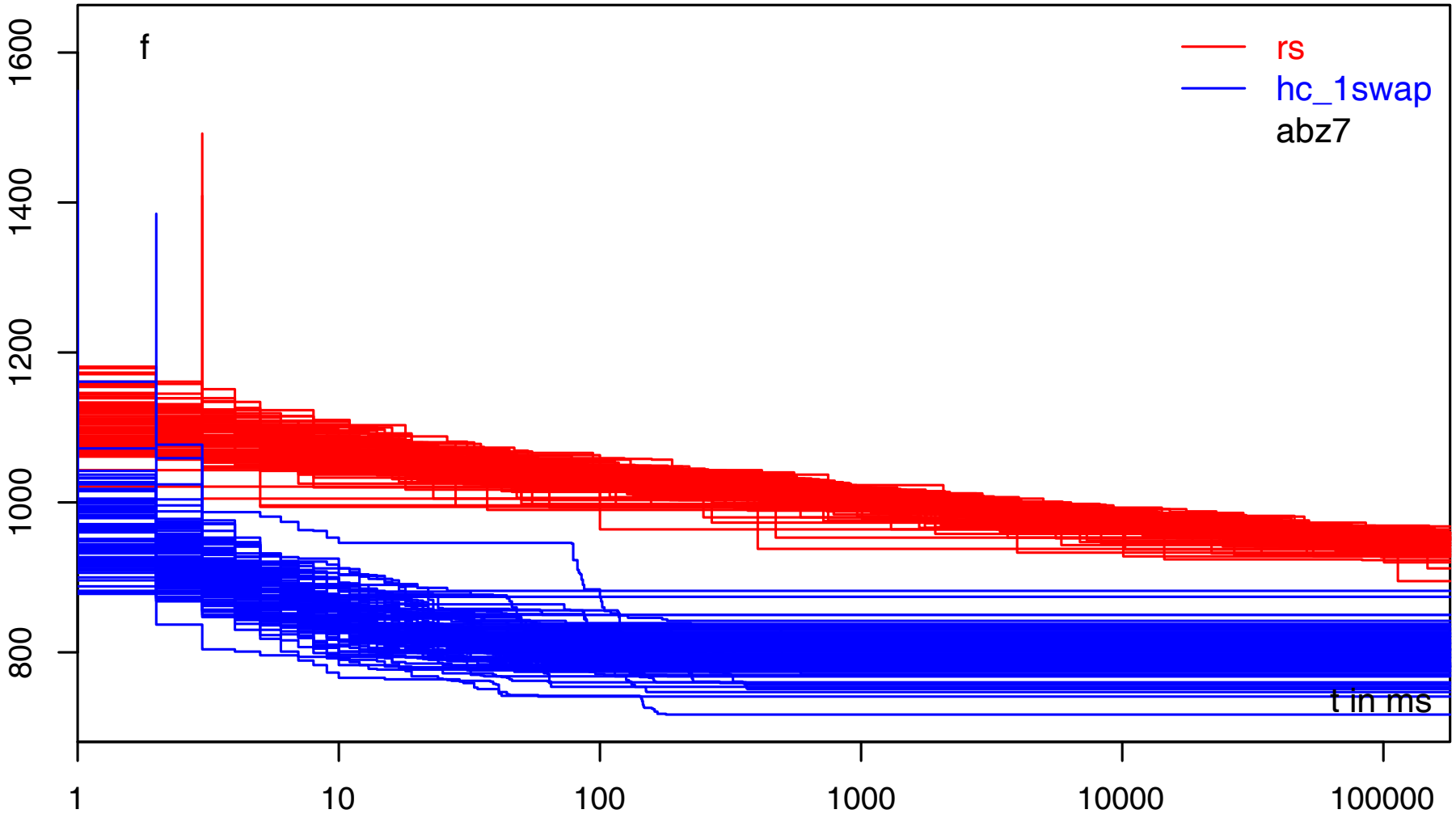


Progress over Time

What progress does the algorithm make over time?

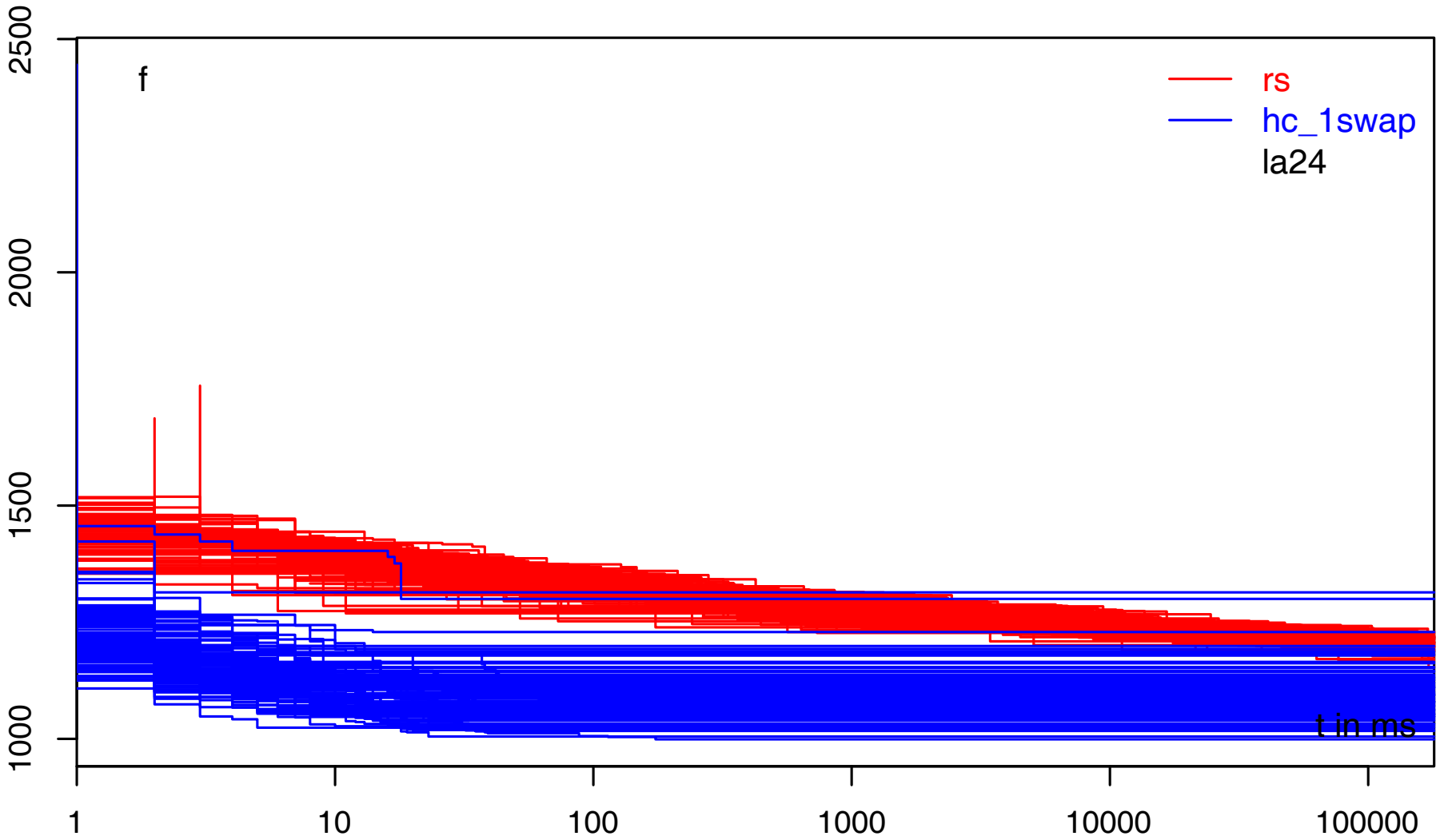
Progress over Time

What progress does the algorithm make over time?



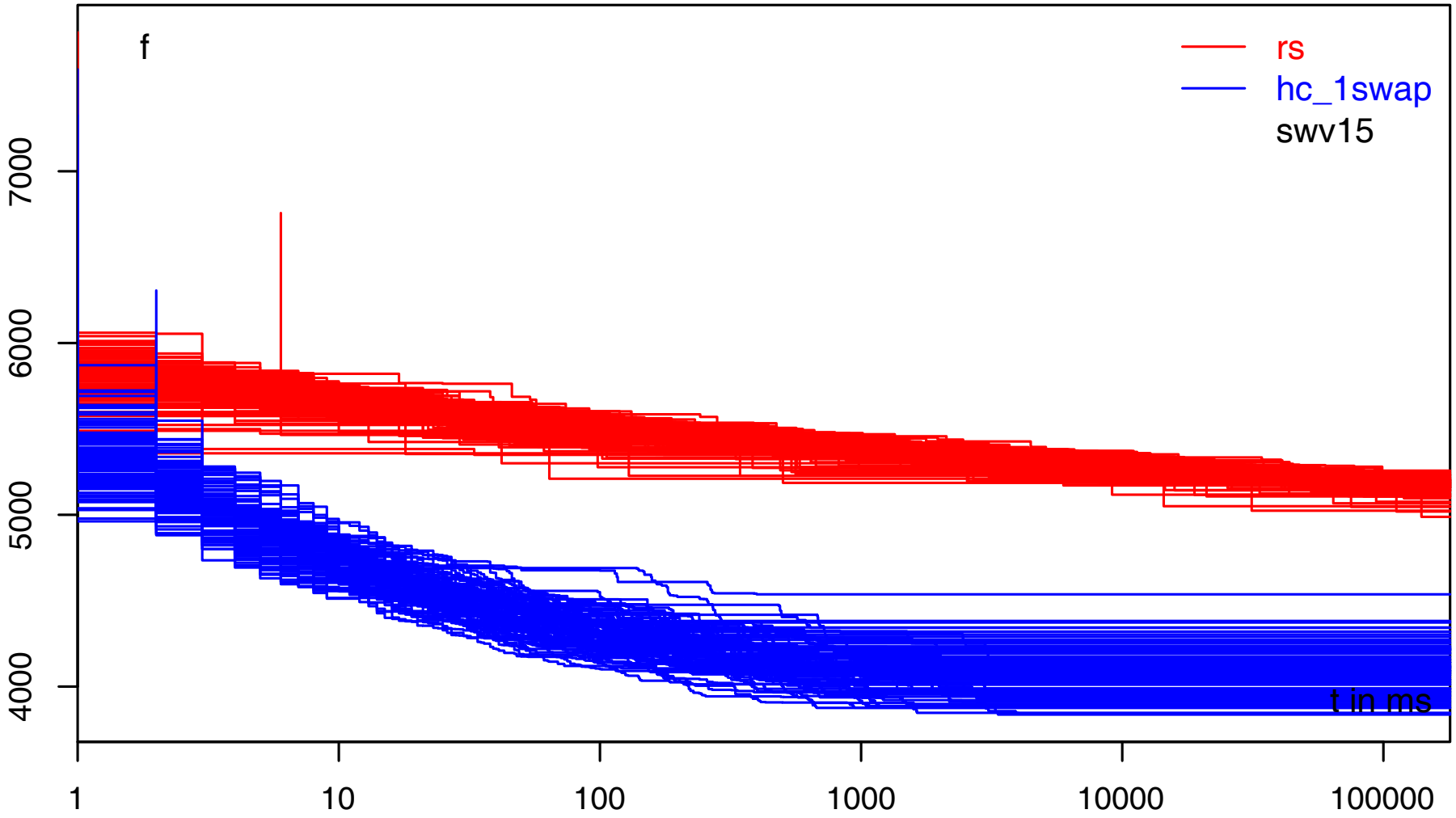
Progress over Time

What progress does the algorithm make over time?



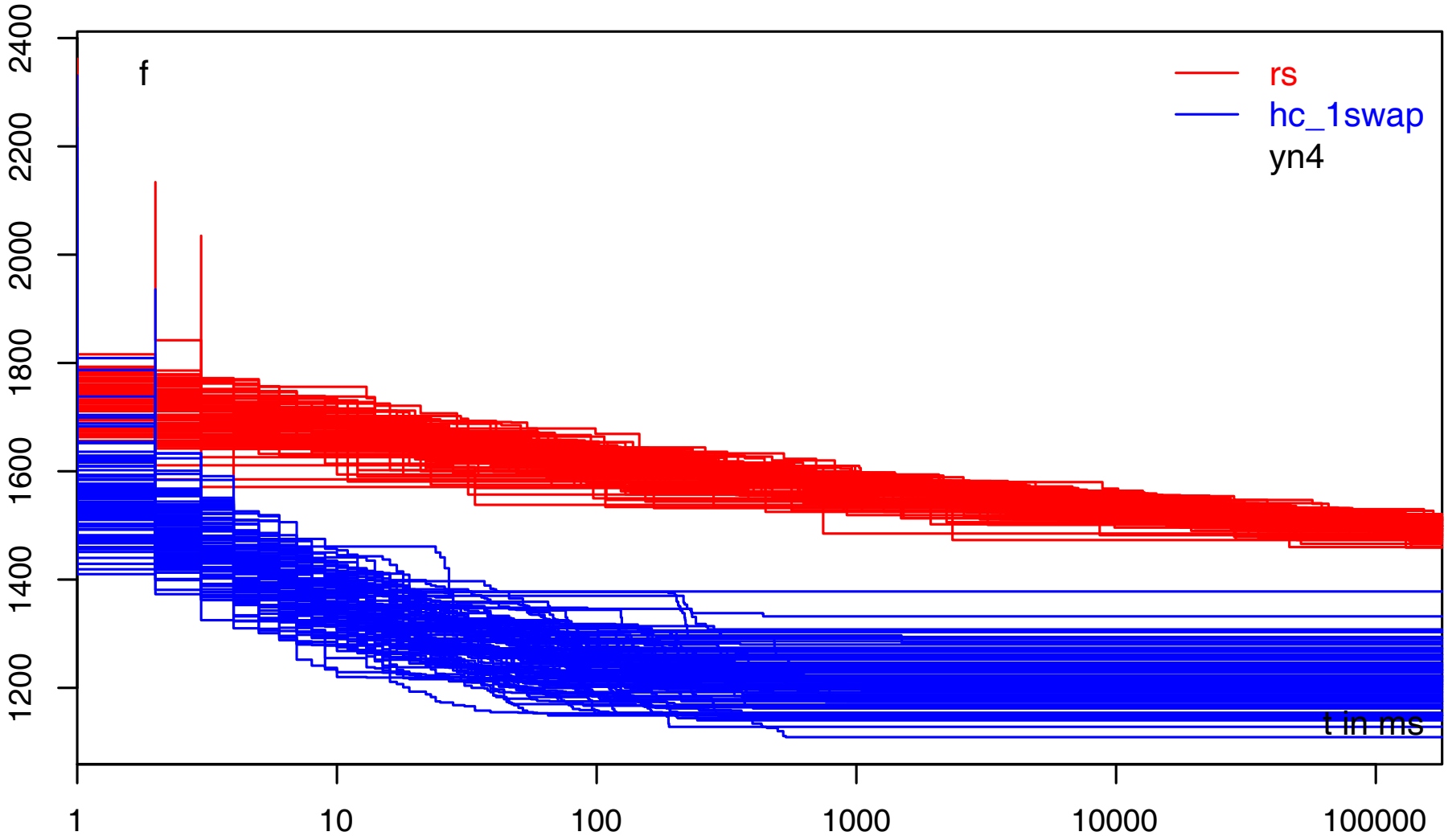
Progress over Time

What progress does the algorithm make over time?



Progress over Time

What progress does the algorithm make over time?

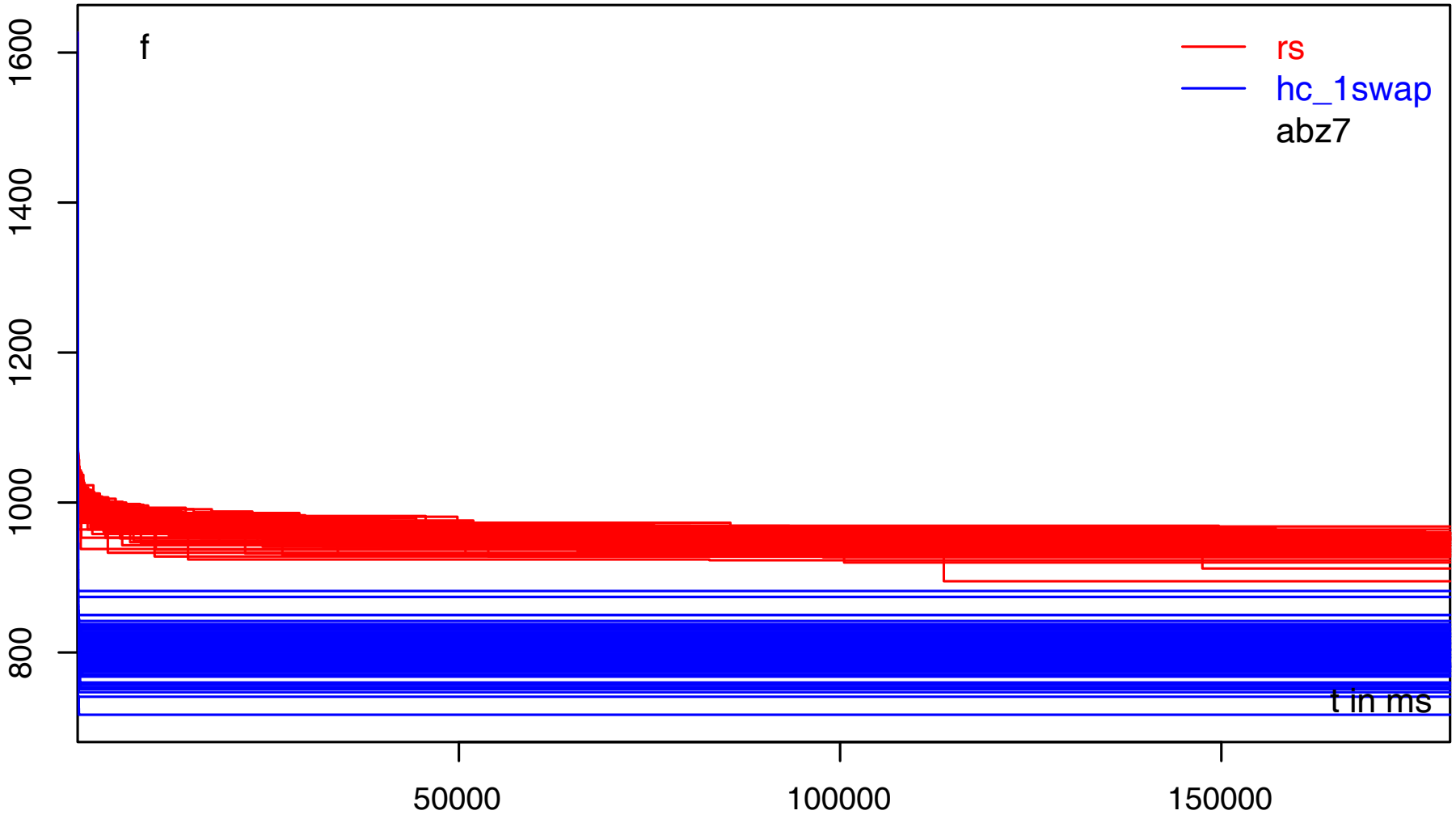


But we waste time...

What if we look at this without log-scaling the time axis?

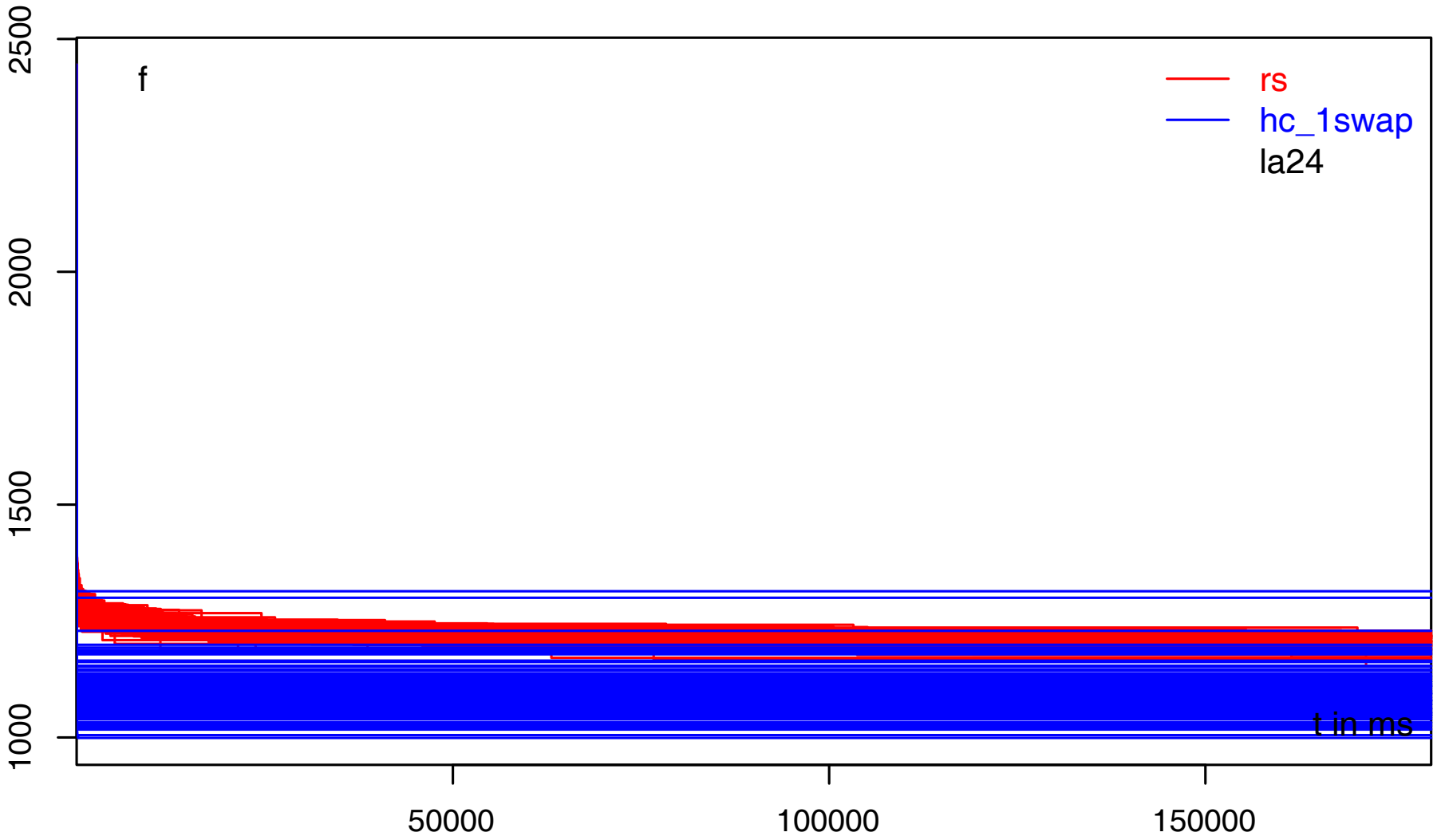
But we waste time...

What if we look at this without log-scaling the time axis?



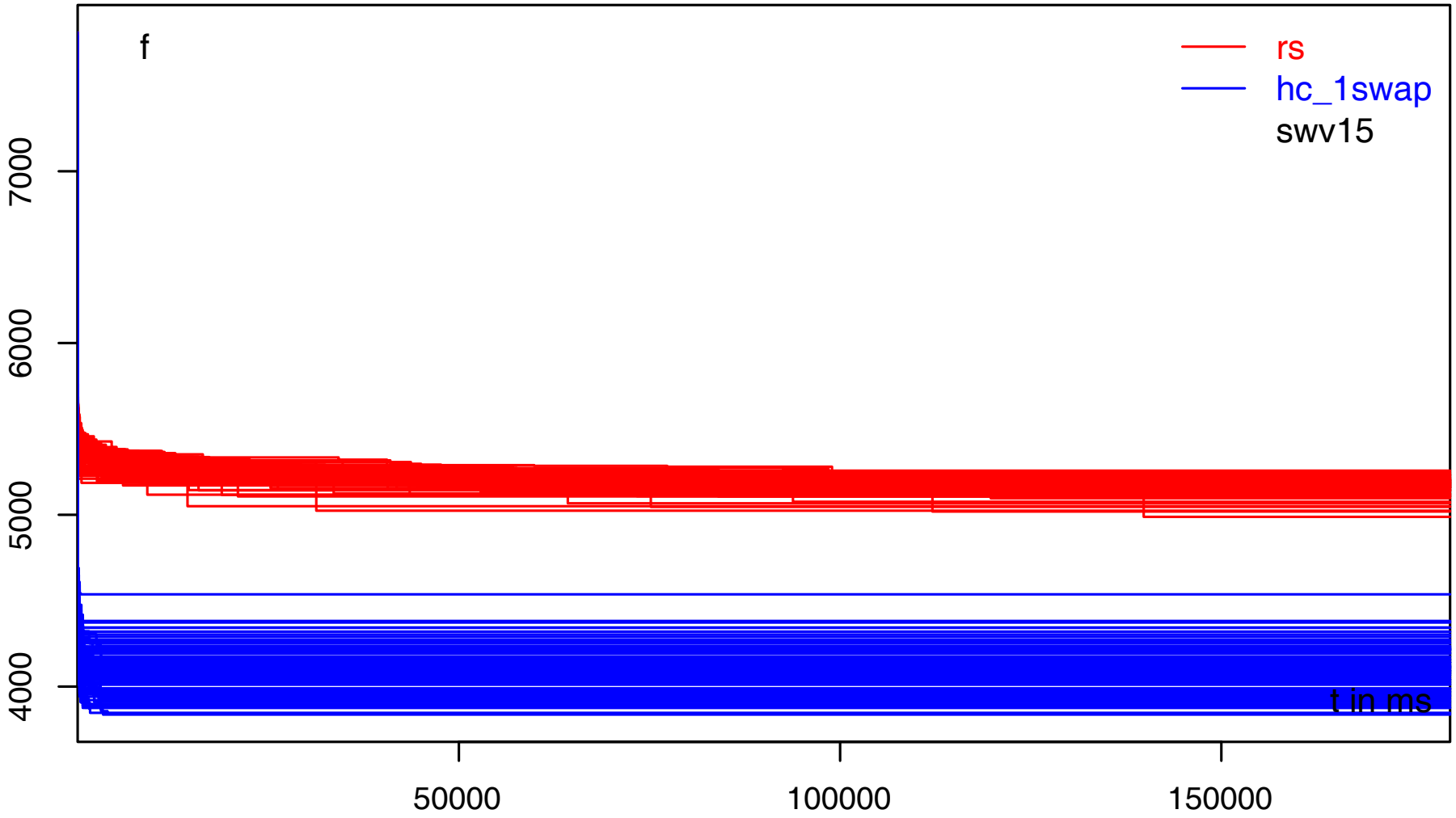
But we waste time...

What if we look at this without log-scaling the time axis?



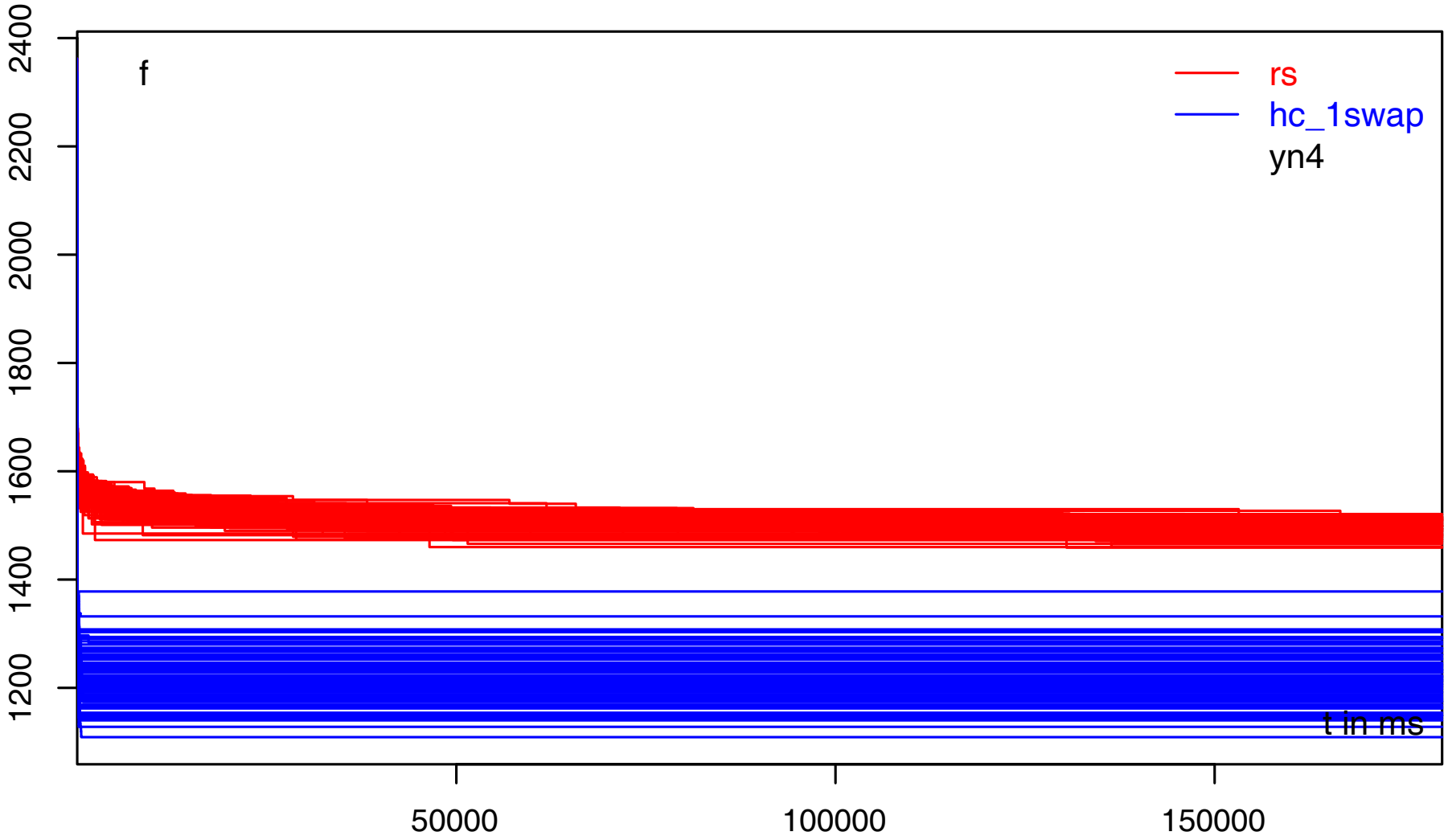
But we waste time...

What if we look at this without log-scaling the time axis?



But we waste time...

What if we look at this without log-scaling the time axis?



But we waste time...

		makespan				last improvement	
$\mathcal{I}$	algo	best	mean	med	sd	med(t)	med(FEs)
abz7	rs	895	945	948	**12**	77s	8'246'019
	hc_1swap	**717**	**800**	**798**	28	0s	16'978
la24	rs	1154	1206	1207	**15**	81s	17'287'329
	hc_1swap	**999**	**1095**	**1086**	56	0s	6612
swv15	rs	4988	5165	5174	**49**	85s	5'525'082
	hc_1swap	**3837**	**4108**	**4108**	137	1s	104'598
yn4	rs	1459	1496	1498	**15**	83s	6'549'694
	hc_1swap	**1109**	**1222**	**1220**	48	0s	31'789

But we waste time...

		makespan				last improvement	
$\mathcal{I}$	algo	best	mean	med	sd	med(t)	med(FEs)
abz7	rs	895	945	948	**12**	77s	8'246'019
	hc_1swap	**717**	**800**	**798**	28	0s	16'978
la24	rs	1154	1206	1207	**15**	81s	17'287'329
	hc_1swap	**999**	**1095**	**1086**	56	0s	6612
swv15	rs	4988	5165	5174	**49**	85s	5'525'082
	hc_1swap	**3837**	**4108**	**4108**	137	1s	104'598
yn4	rs	1459	1496	1498	**15**	83s	6'549'694
	hc_1swap	**1109**	**1222**	**1220**	48	0s	31'789

We have 3min, but our hill climber stops improving after basically 1s!

Premature Convergence

- Our algorithm makes most of its progress early during the search.

Premature Convergence

- Our algorithm makes most of its progress early during the search.
- Later, it basically stagnates and cannot improve.

Premature Convergence

- Our algorithm makes most of its progress early during the search.
- Later, it basically stagnates and cannot improve.
- Why is that?

Premature Convergence

- Our algorithm makes most of its progress early during the search.
- Later, it basically stagnates and cannot improve.
- Why is that?
- The search operator `1swap` defines a neighborhood $N(x) \subset \mathbb{X}$ around a point x.

Premature Convergence

- Our algorithm makes most of its progress early during the search.
- Later, it basically stagnates and cannot improve.
- Why is that?
- The search operator `1swap` defines a neighborhood $N(x) \subset \mathbb{X}$ around a point x.
- The hill climber can only find solutions which are in the neighborhood of the current best solution.

Premature Convergence

- Our algorithm makes most of its progress early during the search.
- Later, it basically stagnates and cannot improve.
- Why is that?
- The search operator `1swap` defines a neighborhood $N(x) \subset \mathbb{X}$ around a point x.
- The hill climber can only find solutions which are in the neighborhood of the current best solution.
- Only the schedules that I can reach by swapping two sub-jobs from two different jobs.

Premature Convergence

- Our algorithm makes most of its progress early during the search.
- Later, it basically stagnates and cannot improve.
- Why is that?
- The search operator `1swap` defines a neighborhood $N(x) \subset \mathbb{X}$ around a point x.
- The hill climber can only find solutions which are in the neighborhood of the current best solution.
- Only the schedules that I can reach by swapping two sub-jobs from two different jobs.
- Clearly $|N(x)| \lll |\mathbb{X}|$!

Premature Convergence

- Our algorithm makes most of its progress early during the search.
- Later, it basically stagnates and cannot improve.
- Why is that?
- The search operator `1swap` defines a neighborhood $N(x) \subset \mathbb{X}$ around a point x.
- The hill climber can only find solutions which are in the neighborhood of the current best solution.
- Only the schedules that I can reach by swapping two sub-jobs from two different jobs.
- Clearly $|N(x)| \lll |\mathbb{X}|$!
- What happens if $f(\gamma(x^\times)) \leq f(\gamma(x)) \forall x \in N(x^\times)$ but $x^\times$ is not the global optimum?

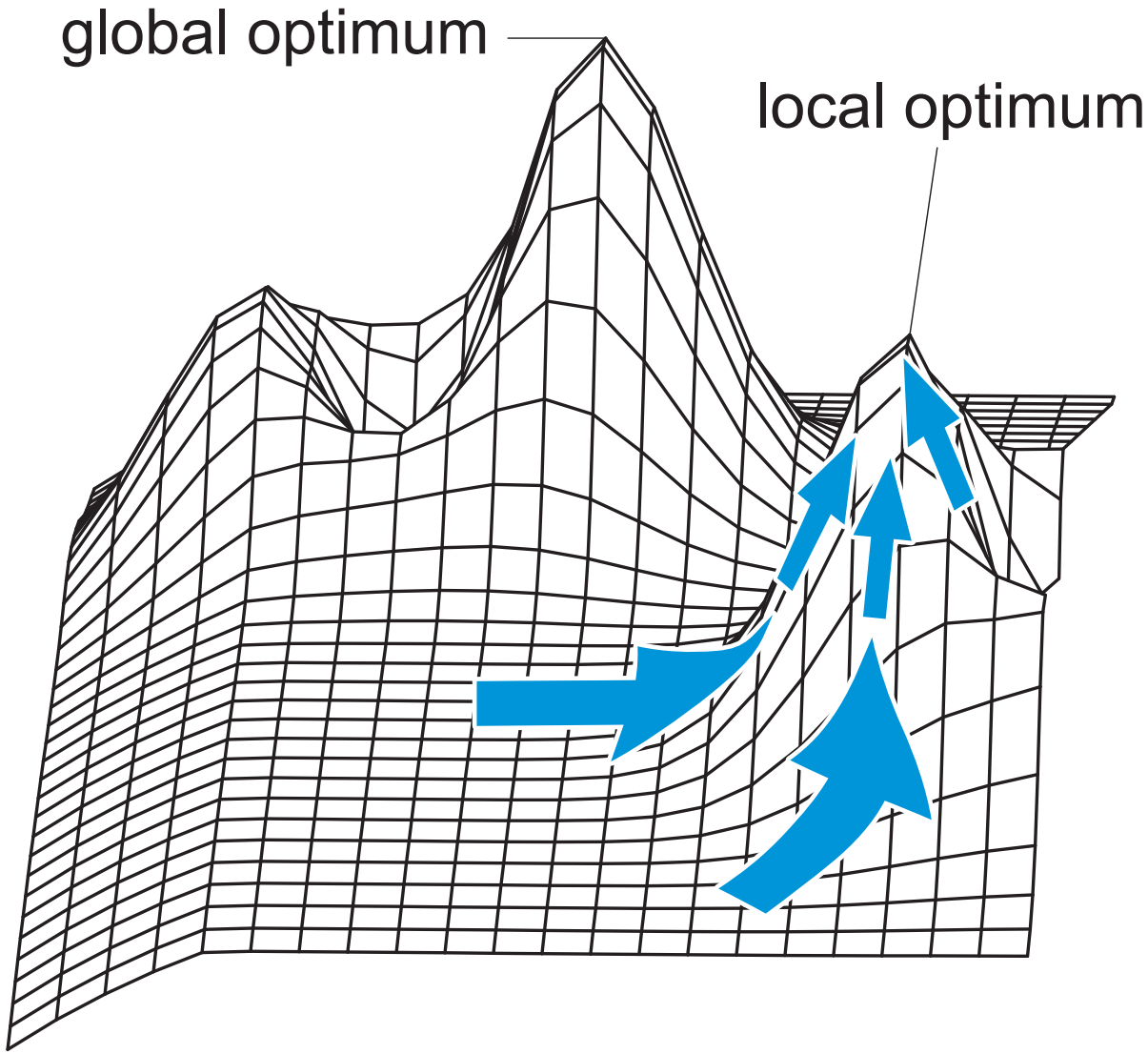
Premature Convergence

- Our algorithm makes most of its progress early during the search.
- Later, it basically stagnates and cannot improve.
- Why is that?
- The search operator `1swap` defines a neighborhood $N(x) \subset \mathbb{X}$ around a point x.
- The hill climber can only find solutions which are in the neighborhood of the current best solution.
- Only the schedules that I can reach by swapping two sub-jobs from two different jobs.
- Clearly $|N(x)| \lll |\mathbb{X}|$!
- What happens if $f(\gamma(x^\times)) \leq f(\gamma(x)) \forall x \in N(x^\times)$ but $x^\times$ is not the global optimum?
- Our algorithm gets trapped in the local optimum $x^\times$ and cannot escape!

Premature Convergence

- Our algorithm makes most of its progress early during the search.
- Later, it basically stagnates and cannot improve.
- Why is that?
- The search operator `1swap` defines a neighborhood $N(x) \subset \mathbb{X}$ around a point x.
- The hill climber can only find solutions which are in the neighborhood of the current best solution.
- Only the schedules that I can reach by swapping two sub-jobs from two different jobs.
- Clearly $|N(x)| \lll |\mathbb{X}|$!
- What happens if $f(\gamma(x^\times)) \leq f(\gamma(x)) \forall x \in N(x^\times)$ but $x^\times$ is not the global optimum?
- Our algorithm gets trapped in the local optimum $x^\times$ and cannot escape!
- This is called Premature Convergence. [8, 9]

Premature Convergence



Section Outline

Stochastic Hill Climber with Restarts

- Idea: We have seen that the results of the hill climber exhibit a relatively high standard deviation.

		makespan				**last improvement**	
$\mathcal{I}$	algo	best	mean	med	sd	med(t)	med(FEs)
abz7	rs	895	945	948	**12**	77s	8'246'019
	hc_1swap	**717**	**800**	**798**	28	0s	16'978
la24	rs	1154	1206	1207	**15**	81s	17'287'329
	hc_1swap	**999**	**1095**	**1086**	56	0s	6612
swv15	rs	4988	5165	5174	**49**	85s	5'525'082
	hc_1swap	**3837**	**4108**	**4108**	137	1s	104'598
yn4	rs	1459	1496	1498	**15**	83s	6'549'694
	hc_1swap	**1109**	**1222**	**1220**	48	0s	31'789

Stochastic Hill Climber with Restarts

- Idea: We have seen that the results of the hill climber exhibit a relatively high standard deviation.
- At the same time, a single *run* of the algorithm converges quickly.

		makespan				**last improvement**	
$\mathcal{I}$	algo	best	mean	med	sd	med(t)	med(FEs)
abz7	rs	895	945	948	**12**	77s	8'246'019
	hc_1swap	**717**	**800**	**798**	28	0s	16'978
la24	rs	1154	1206	1207	**15**	81s	17'287'329
	hc_1swap	**999**	**1095**	**1086**	56	0s	6612
swv15	rs	4988	5165	5174	**49**	85s	5'525'082
	hc_1swap	**3837**	**4108**	**4108**	137	1s	104'598
yn4	rs	1459	1496	1498	**15**	83s	6'549'694
	hc_1swap	**1109**	**1222**	**1220**	48	0s	31'789

Stochastic Hill Climber with Restarts

- Idea: We have seen that the results of the hill climber exhibit a relatively high standard deviation.
- At the same time, a single *run* of the algorithm converges quickly.
- Let us exploit this variation!

Stochastic Hill Climber with Restarts

- Idea: We have seen that the results of the hill climber exhibit a relatively high standard deviation.
- At the same time, a single *run* of the algorithm converges quickly.
- Let us exploit this variation!
- Idea: If we did not make any progress for some time t, we simply restart at a new random solution.

Stochastic Hill Climber with Restarts

- Idea: We have seen that the results of the hill climber exhibit a relatively high standard deviation.
- At the same time, a single *run* of the algorithm converges quickly.
- Let us exploit this variation!
- Idea: If we did not make any progress for some time t, we simply restart at a new random solution.
- Of course, we will always remember the overall best solution we ever had (in another variable).

Stochastic Hill Climber with Restarts

- Idea: We have seen that the results of the hill climber exhibit a relatively high standard deviation.
- At the same time, a single *run* of the algorithm converges quickly.
- Let us exploit this variation!
- Idea: If we did not make any progress for some time t, we simply restart at a new random solution.
- Of course, we will always remember the overall best solution we ever had (in another variable).
- Since we do not know which value of t is good, start small and increase it after each restart.

So what do we get?

- I execute the program 101 times for each of the datasets `abz7`, `la24`, `swv15`, and `yn4`

So what do we get?

- I execute the program 101 times for each of the datasets abz7, la24, swv15, and yn4

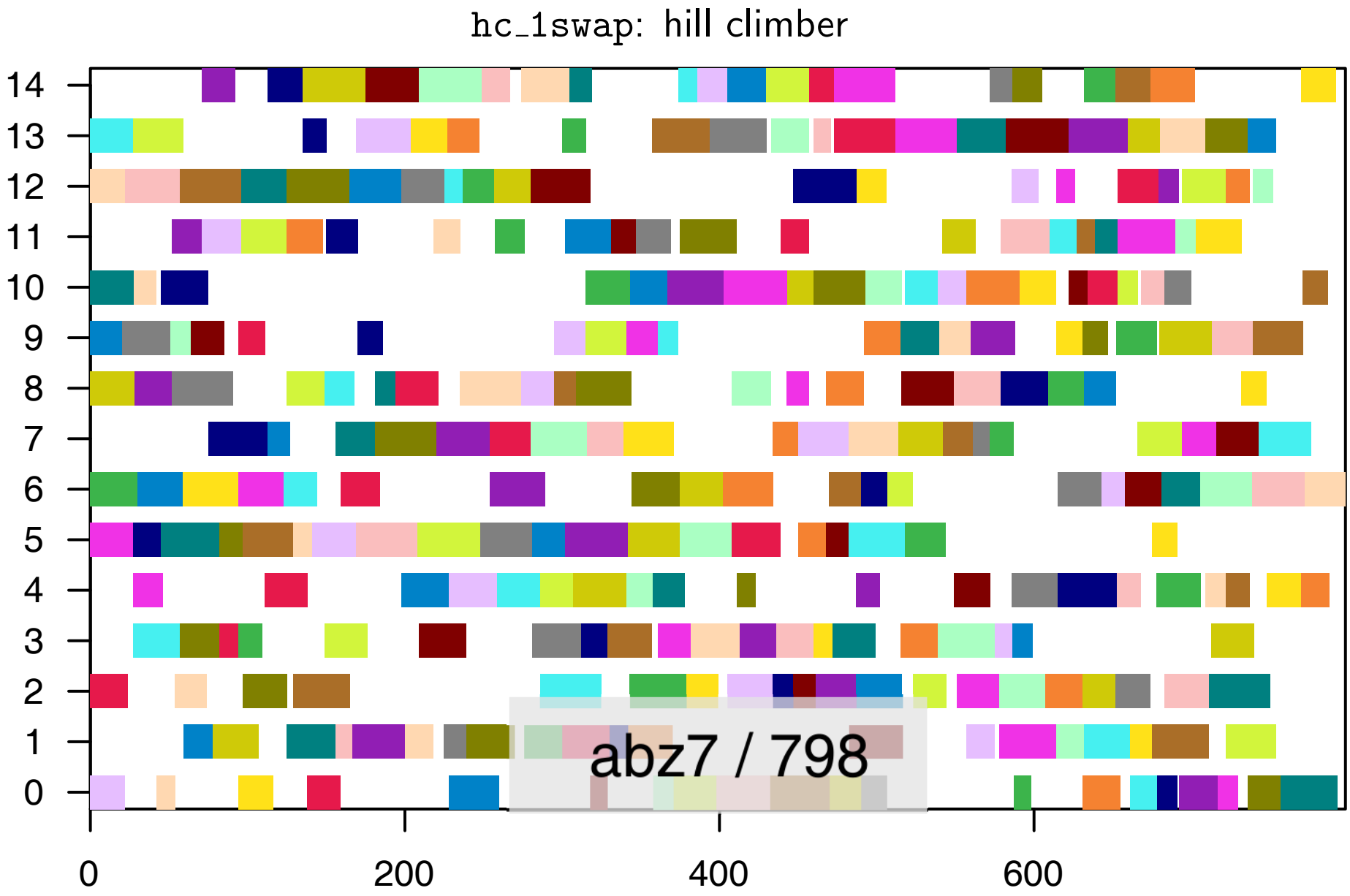
		makespan				last improvement	
$\mathcal{I}$	algo	best	mean	med	sd	med(t)	med(FEs)
abz7	rs	895	945	948	12	77s	8'246'019
	hc_1swap	**717**	800	798	28	0s	16'978
	hcr_256+5%_1swap	723	**742**	**743**	**7**	21s	5'681'591
la24	rs	1154	1206	1207	15	81s	17'287'329
	hc_1swap	999	1095	1086	56	0s	6612
	hcr_256+5%_1swap	**970**	**997**	**998**	**9**	6s	3'470'368
swv15	rs	4988	5165e	5174	49	85s	5'525'082
	hc_1swap	3837	4108	4108	137	1s	104'598
	hcr_256+5%_1swap	**3701**	**3850**	**3857**	**40**	60s	9'874'102
yn4	rs	1459	1496	1498	15	83s	6'549'694
	hc_1swap	1109	1222	1220	48	0s	31'789
	hcr_256+5%_1swap	**1095**	**1129**	**1130**	**14**	22s	4'676'669

So what do we get?

- It is still the *same* algorithm. Restarting will not always make it better (e.g., if I restart too early) – but it will often do.

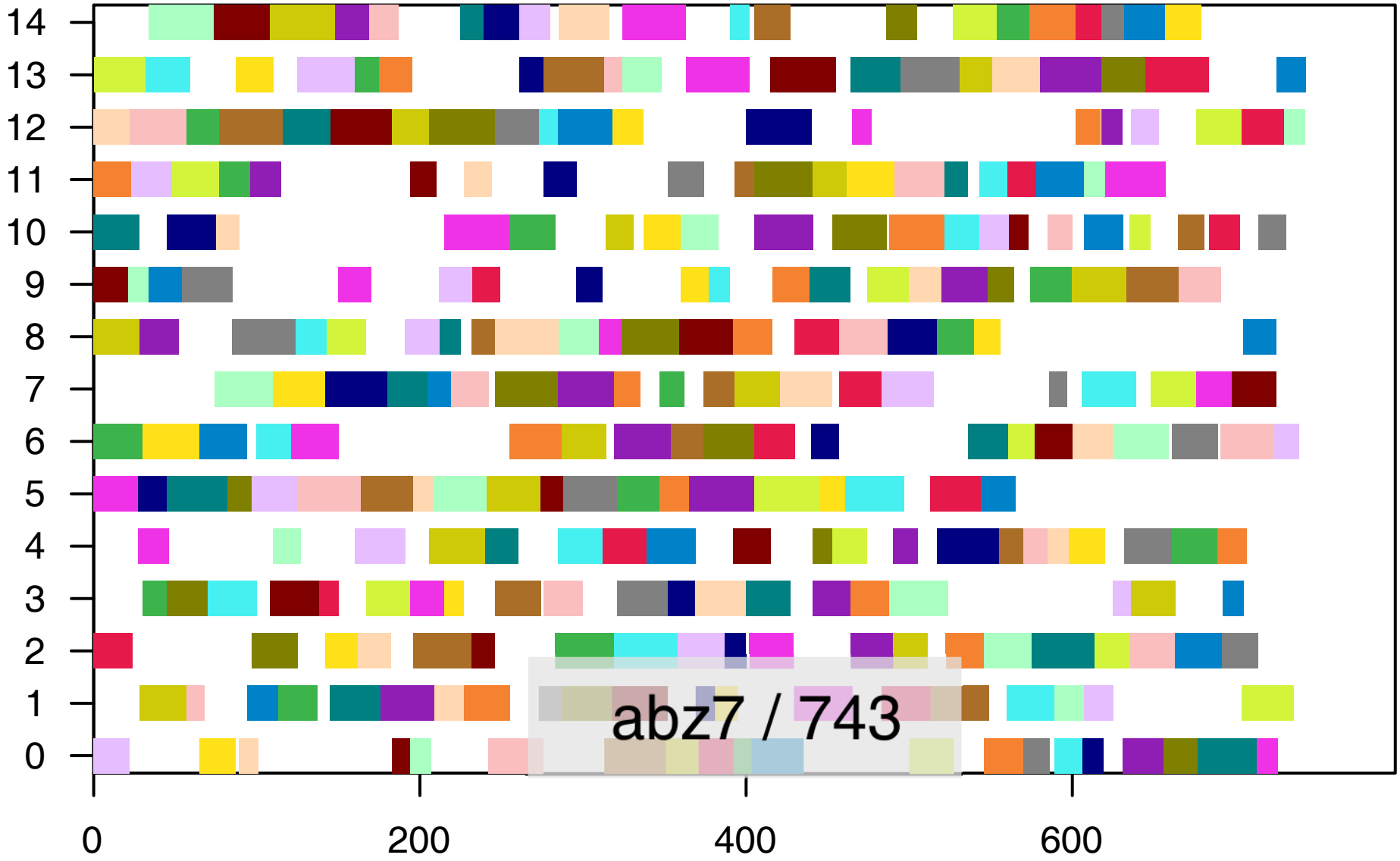
		makespan				last improvement	
$\mathcal{I}$	algo	best	mean	med	sd	med(t)	med(FEs)
abz7	rs	895	945	948	12	77s	8'246'019
	hc_1swap	**717**	800	798	28	0s	16'978
	hcr_256+5%_1swap	723	**742**	**743**	**7**	21s	5'681'591
la24	rs	1154	1206	1207	15	81s	17'287'329
	hc_1swap	999	1095	1086	56	0s	6612
	hcr_256+5%_1swap	**970**	**997**	**998**	**9**	6s	3'470'368
swv15	rs	4988	5165e	5174	49	85s	5'525'082
	hc_1swap	3837	4108	4108	137	1s	104'598
	hcr_256+5%_1swap	**3701**	**3850**	**3857**	**40**	60s	9'874'102
yn4	rs	1459	1496	1498	15	83s	6'549'694
	hc_1swap	1109	1222	1220	48	0s	31'789
	hcr_256+5%_1swap	**1095**	**1129**	**1130**	**14**	22s	4'676'669

So what do we get?

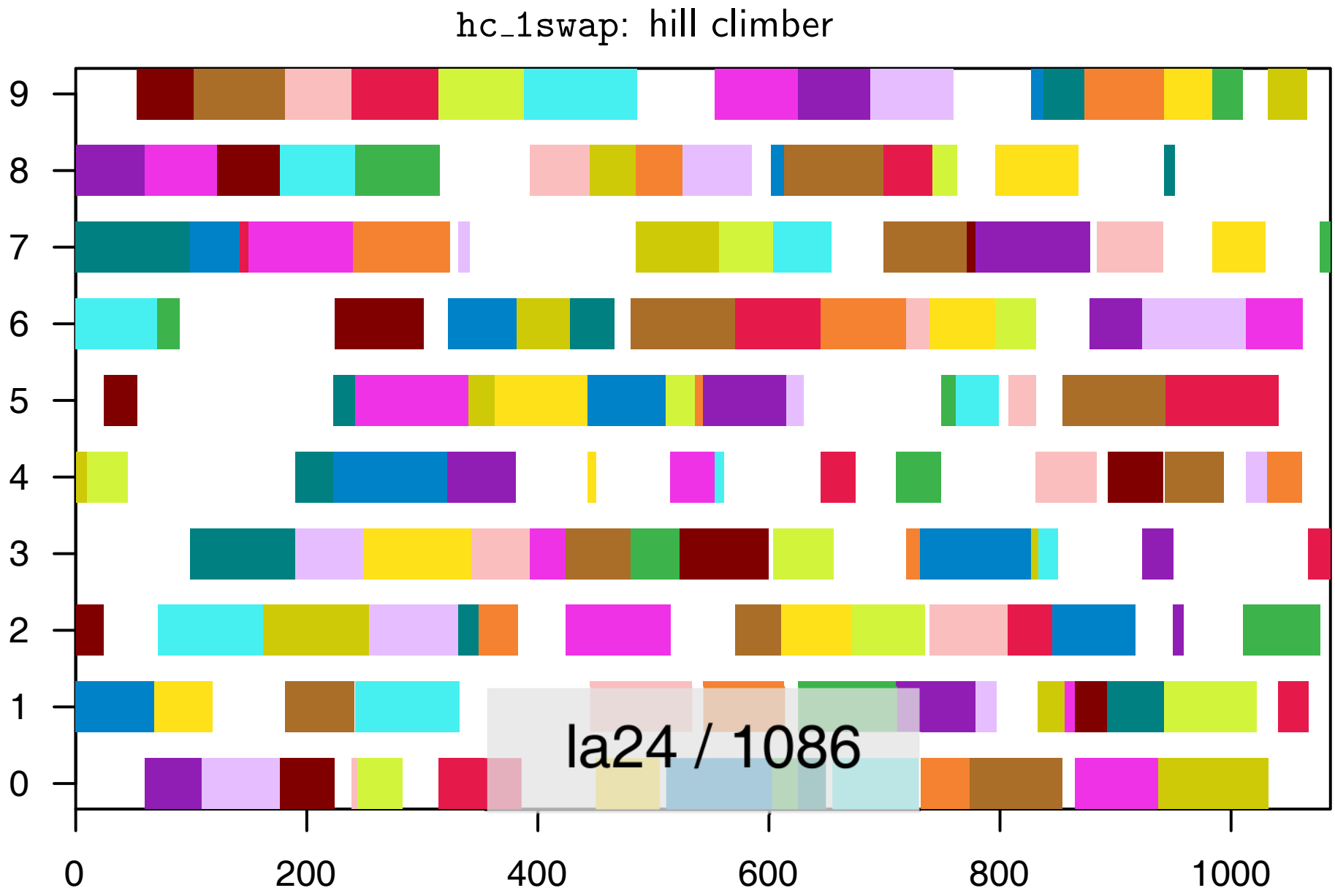


So what do we get?

`hcr_256+5%_1swap`: hill climber with restarts after 256+5% non-improv steps

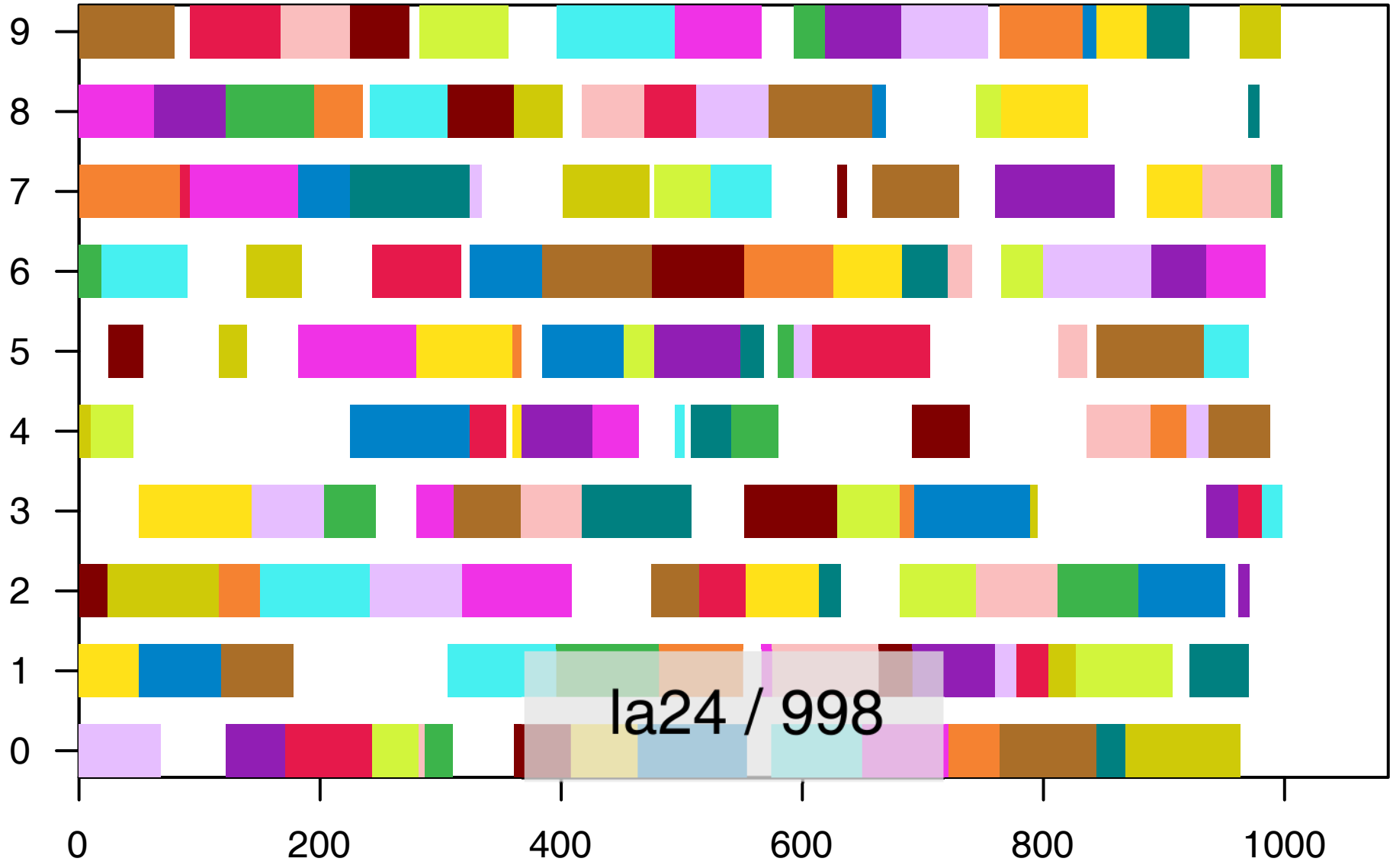


So what do we get?

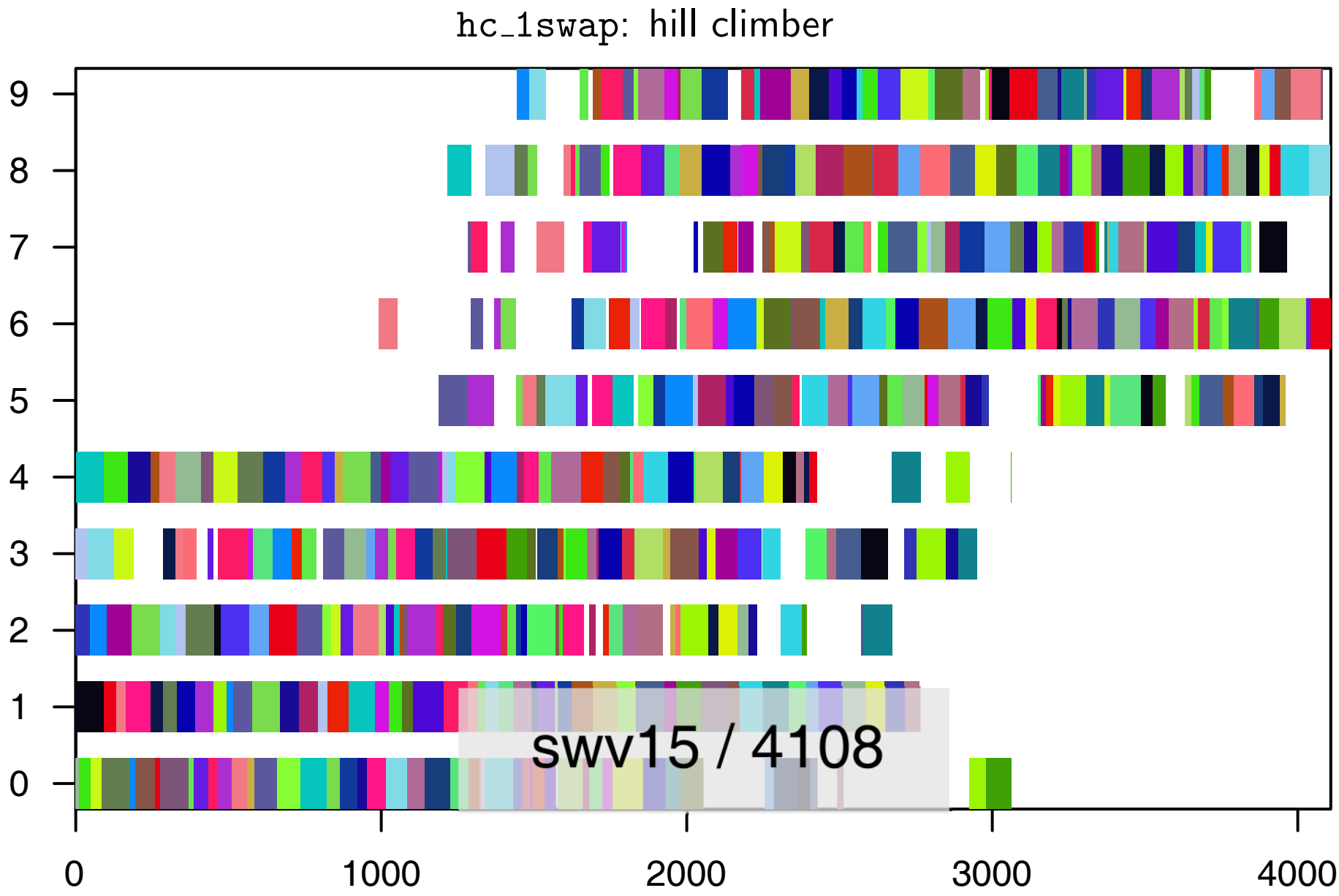


So what do we get?

`hcr_256+5%_1swap`: hill climber with restarts after 256+5% non-improv steps

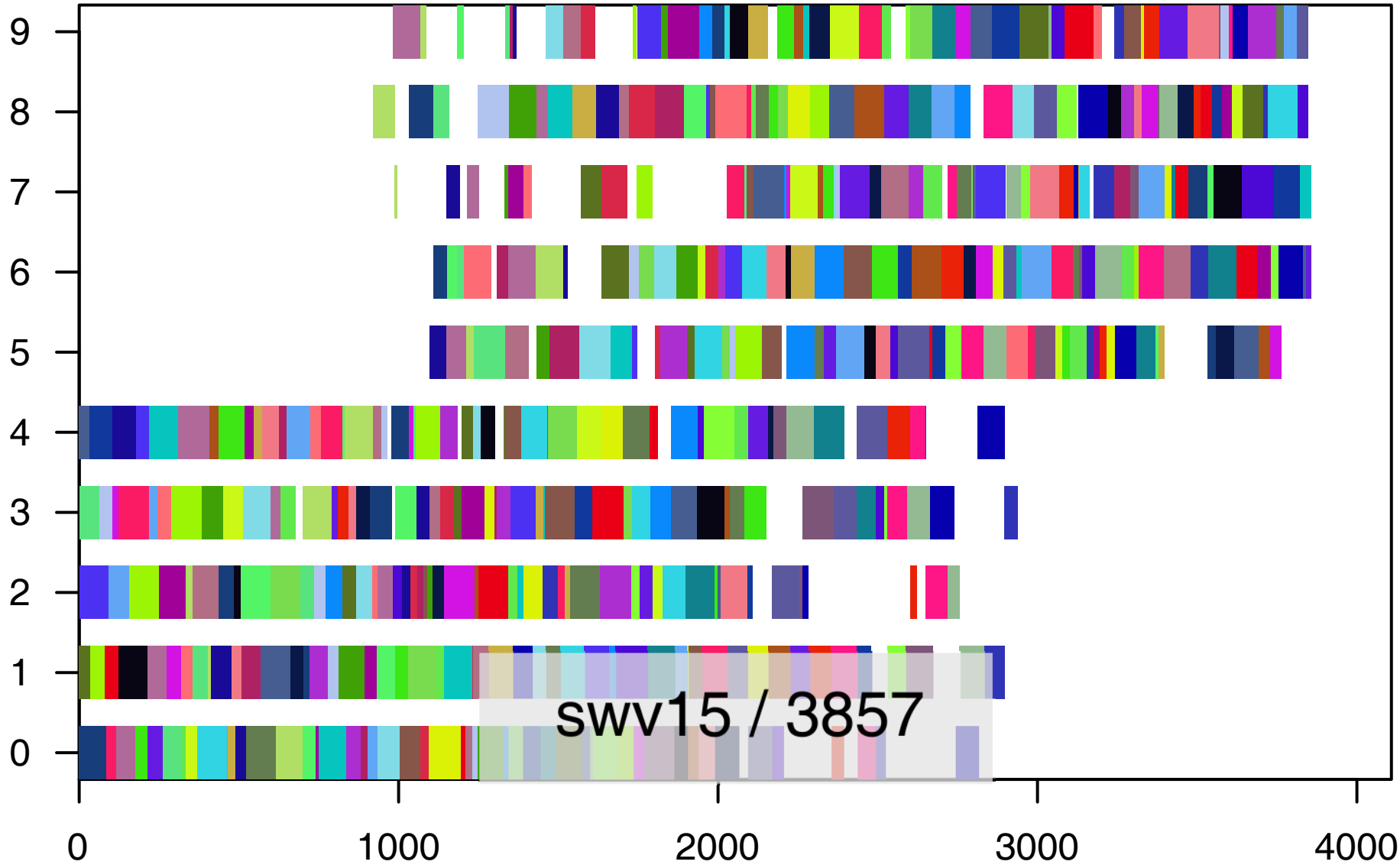


So what do we get?

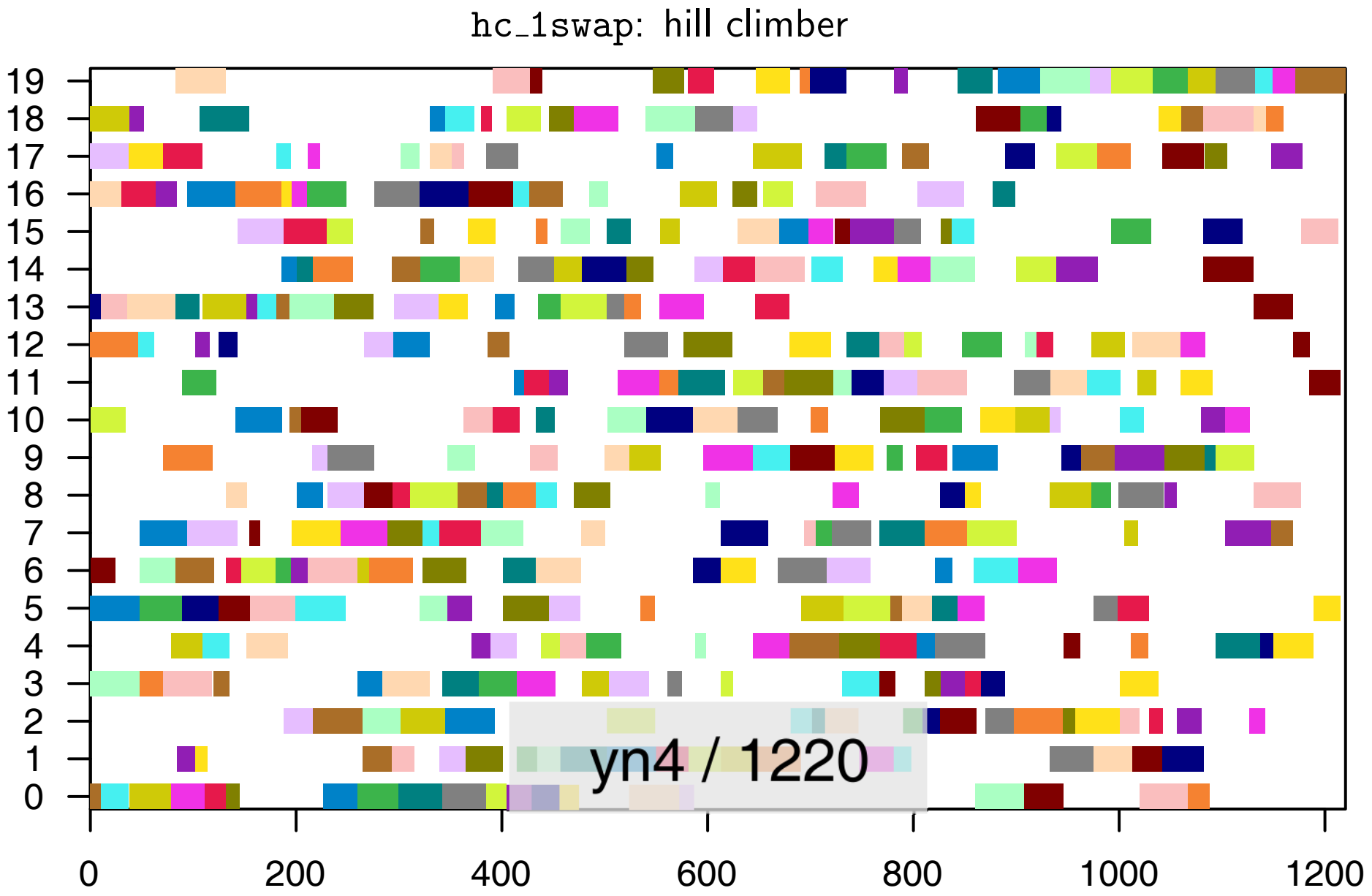


So what do we get?

`hcr_256+5%_1swap`: hill climber with restarts after 256+5% non-improv steps

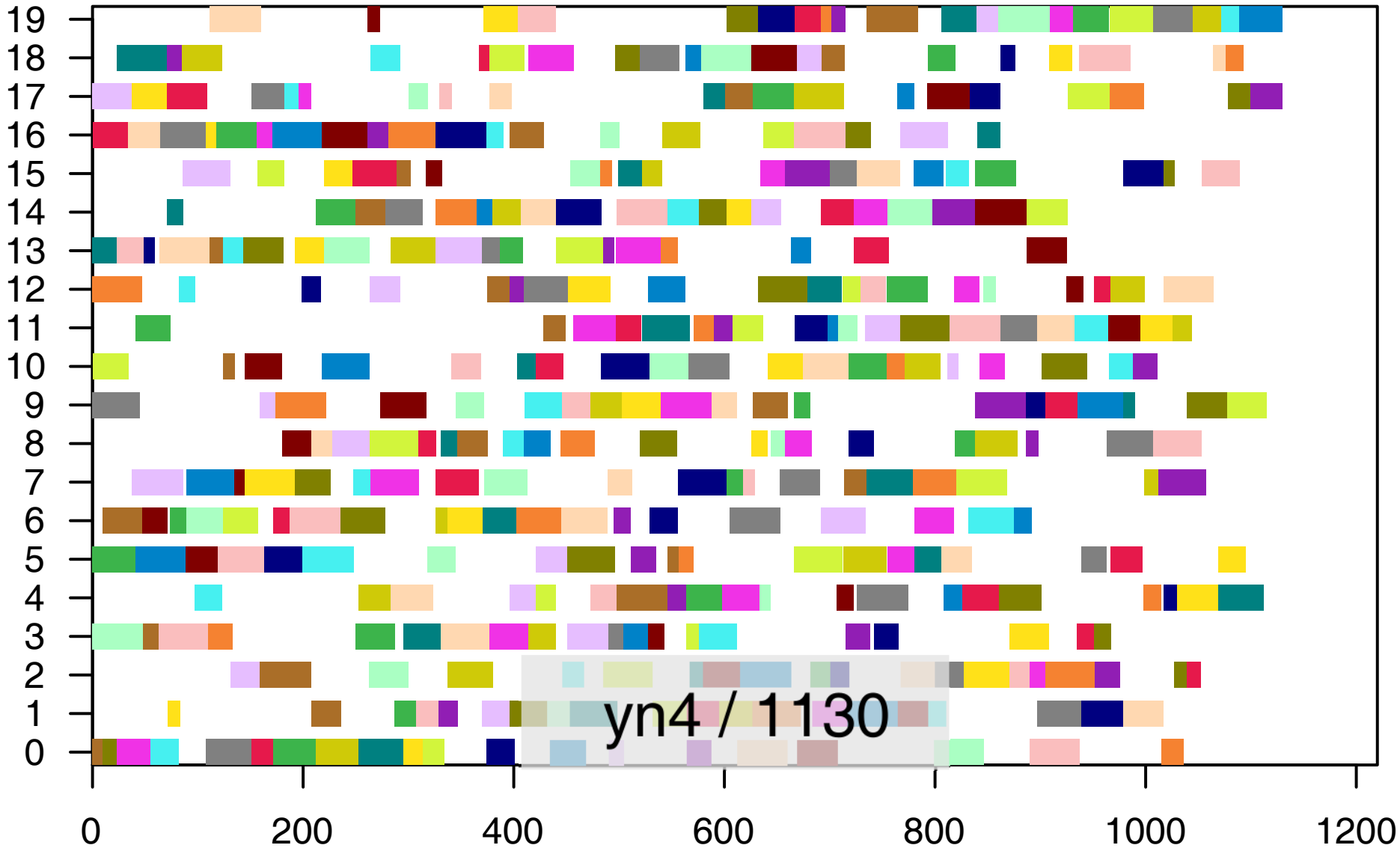


So what do we get?



So what do we get?

`hcr_256+5%_1swap`: hill climber with restarts after 256+5% non-improv steps

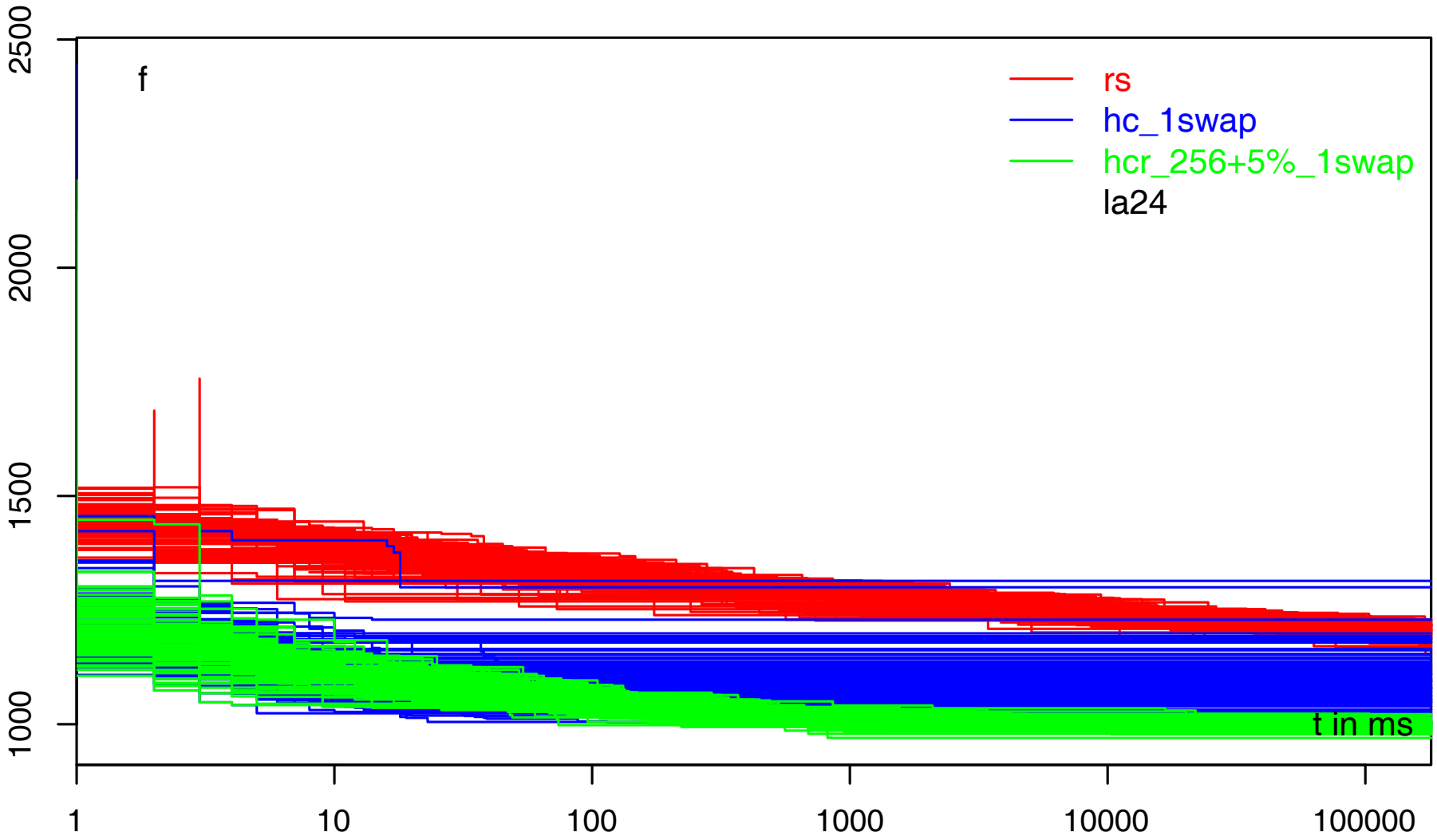


Progress over Time

What progress does the algorithm make over time?

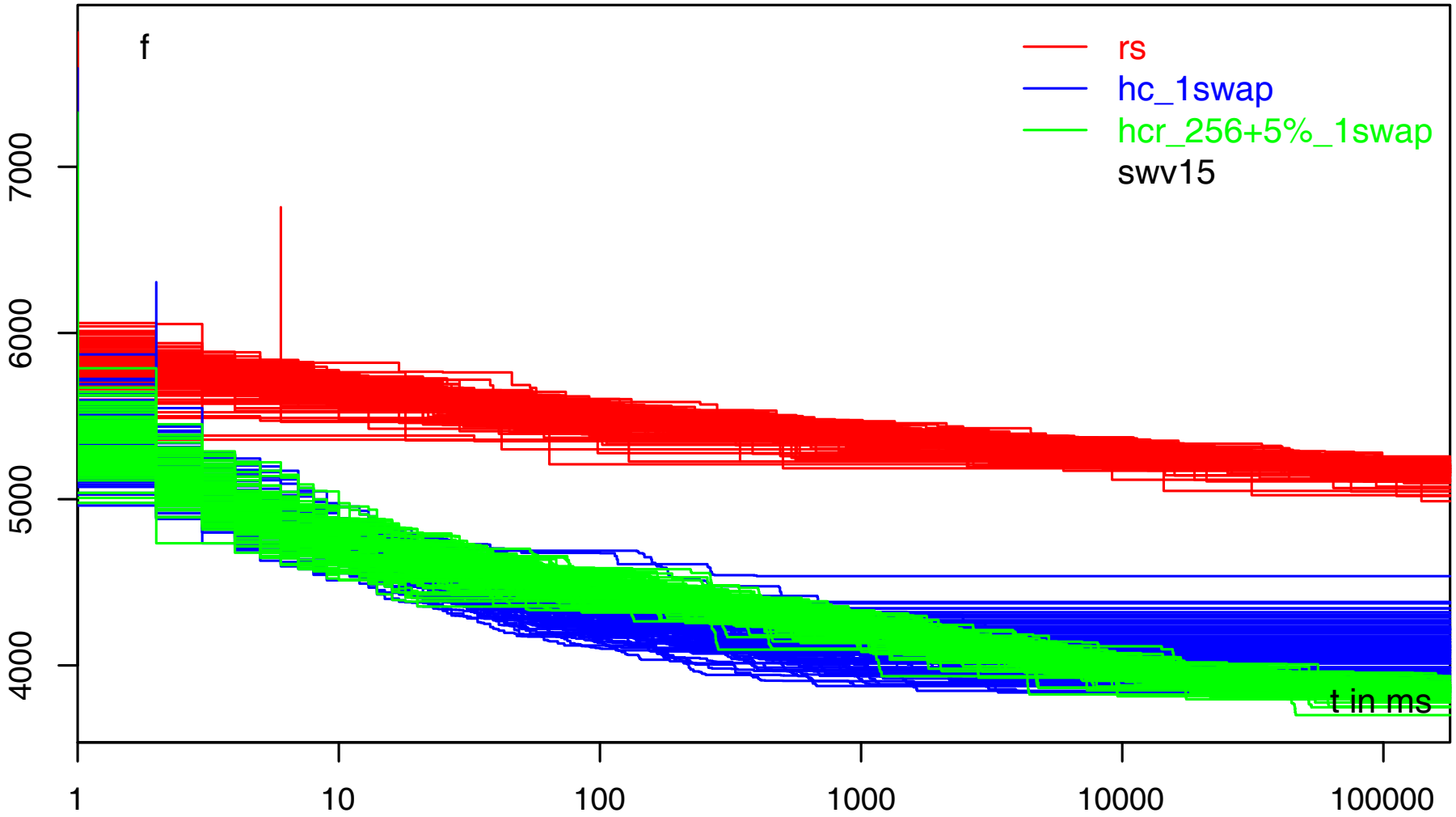
Progress over Time

What progress does the algorithm make over time?



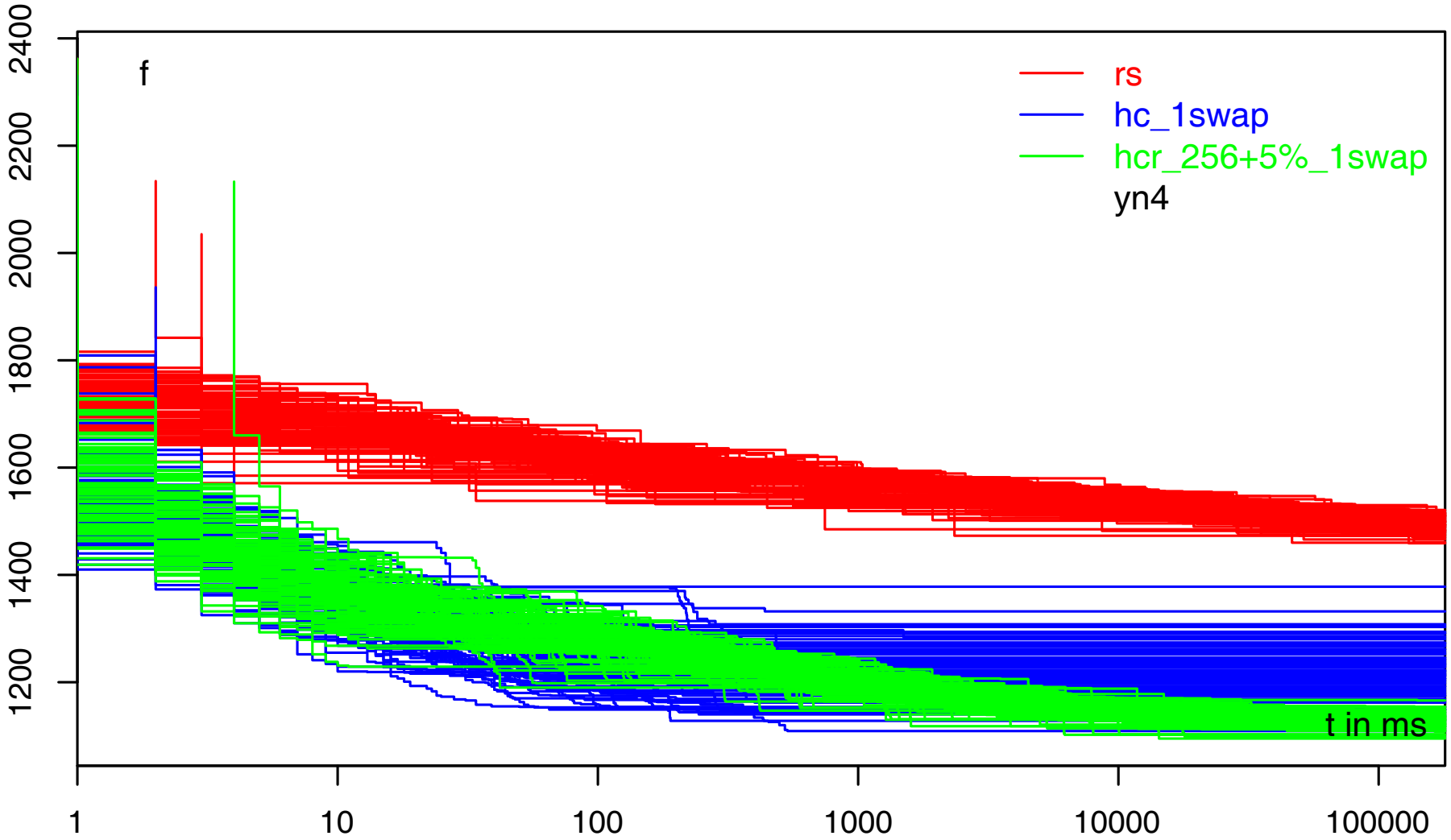
Progress over Time

What progress does the algorithm make over time?



Progress over Time

What progress does the algorithm make over time?



Section Outline

How else can we stop premature convergence?

- Our (original) hill climber will stop improving if it can no longer finder better solutions.

How else can we stop premature convergence?

- Our (original) hill climber will stop improving if it can no longer finder better solutions.
- This happens when it reaches a local optimum.

How else can we stop premature convergence?

- Our (original) hill climber will stop improving if it can no longer finder better solutions.
- This happens when it reaches a local optimum.
- A local optimum is a point $x^\times$ in $\mathbb{X}$ where no `1swap`-move can yield any improvement.

How else can we stop premature convergence?

- Our (original) hill climber will stop improving if it can no longer finder better solutions.
- This happens when it reaches a local optimum.
- A local optimum is a point $x^\times$ in $\mathbb{X}$ where no `1swap`-move can yield any improvement.
- It does not matter which two job ids I exchange in the current best string $x^\times$, the result is not better than $x^\times$.

How else can we stop premature convergence?

- Our (original) hill climber will stop improving if it can no longer finder better solutions.
- This happens when it reaches a local optimum.
- A local optimum is a point $x^\times$ in $\mathbb{X}$ where no `1swap`-move can yield any improvement.
- It does not matter which two job ids I exchange in the current best string $x^\times$, the result is not better than $x^\times$.
- Notice: Which a local optimum is, is determined by the unary search operator!

How else can we stop premature convergence?

- Our (original) hill climber will stop improving if it can no longer finder better solutions.
- This happens when it reaches a local optimum.
- A local optimum is a point $x^\times$ in $\mathbb{X}$ where no `1swap`-move can yield any improvement.
- It does not matter which two job ids I exchange in the current best string $x^\times$, the result is not better than $x^\times$.
- Notice: Which a local optimum is, is determined by the unary search operator!
- If we had a different operator with a bigger neighborhood, then maybe $x^\times$ would no longer be a local optimum and we could still improve the results after reaching it...

Making the neighborhood bigger

- Two solutions x_1 and x_2 are "neighbors" if I can reach x_2 by applying the search operator one time to x_1.

Making the neighborhood bigger

- Two solutions x_1 and x_2 are "neighbors" if I can reach x_2 by applying the search operator one time to x_1.
- The search operator determines which solutions are "neighbors".

Making the neighborhood bigger

- Two solutions x_1 and x_2 are "neighbors" if I can reach x_2 by applying the search operator one time to x_1.
- The search operator determines which solutions are "neighbors".
- The neighborhood determines what a local optimum is.

Making the neighborhood bigger

- Two solutions x_1 and x_2 are "neighbors" if I can reach x_2 by applying the search operator one time to x_1.
- The search operator determines which solutions are "neighbors".
- The neighborhood determines what a local optimum is.
- Let's make it bigger.

Making the neighborhood bigger

- Two solutions x_1 and x_2 are "neighbors" if I can reach x_2 by applying the search operator one time to x_1.
- The search operator determines which solutions are "neighbors".
- The neighborhood determines what a local optimum is.
- Let's make it bigger.
- We could swap more than 2 job ids!

Making the neighborhood bigger

- Two solutions x_1 and x_2 are "neighbors" if I can reach x_2 by applying the search operator one time to x_1.
- The search operator determines which solutions are "neighbors".
- The neighborhood determines what a local optimum is.
- Let's make it bigger.
- We could swap more than 2 job ids!
- But we should respect the causality: small changes to the solution cause small changes in the objective value – big changes will lead to unpredictable results.

Making the neighborhood bigger

- Two solutions x_1 and x_2 are "neighbors" if I can reach x_2 by applying the search operator one time to x_1.
- The search operator determines which solutions are "neighbors".
- The neighborhood determines what a local optimum is.
- Let's make it bigger.
- We could swap more than 2 job ids!
- But we should respect the causality: small changes to the solution cause small changes in the objective value – big changes will lead to unpredictable results.
- If we just change everything always, we basically have random sampling again...

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 25% in total), we swap 3 job ids and quit.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 25% in total), we swap 3 job ids and quit.
  3. otherwise (it was tail), we again flip a coin.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 25% in total), we swap 3 job ids and quit.
  3. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 12.5% in total), we swap 4 job ids and quit.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 25% in total), we swap 3 job ids and quit.
  3. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 12.5% in total), we swap 4 job ids and quit.
  4. otherwise (it was tail), we again flip a coin.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 25% in total), we swap 3 job ids and quit.
  3. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 12.5% in total), we swap 4 job ids and quit.
  4. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 6.25% in total), we swap 5 job ids and quit.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 25% in total), we swap 3 job ids and quit.
  3. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 12.5% in total), we swap 4 job ids and quit.
  4. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 6.25% in total), we swap 5 job ids and quit.
  5. otherwise (it was tail), we again flip a coin.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 25% in total), we swap 3 job ids and quit.
  3. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 12.5% in total), we swap 4 job ids and quit.
  4. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 6.25% in total), we swap 5 job ids and quit.
  5. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 3.125% in total), we swap 6 job ids and quit.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 25% in total), we swap 3 job ids and quit.
  3. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 12.5% in total), we swap 4 job ids and quit.
  4. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 6.25% in total), we swap 5 job ids and quit.
  5. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 3.125% in total), we swap 6 job ids and quit.
  6. and so on.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 25% in total), we swap 3 job ids and quit.
  3. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 12.5% in total), we swap 4 job ids and quit.
  4. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 6.25% in total), we swap 5 job ids and quit.
  5. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 3.125% in total), we swap 6 job ids and quit.
  6. and so on.
- We most often make small moves, but sometimes bigger ones.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 25% in total), we swap 3 job ids and quit.
  3. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 12.5% in total), we swap 4 job ids and quit.
  4. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 6.25% in total), we swap 5 job ids and quit.
  5. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 3.125% in total), we swap 6 job ids and quit.
  6. and so on.
- We most often make small moves, but sometimes bigger ones.
- Thereotically, we could always escape from local any optima

.

Making the neighborhood bigger

- Idea: Let's most often swap 2 jobs, but sometimes 3, less often 4, from time to time 5, rarely 6, hardly ever 7, ...
- `nswap` operator idea:
  1. flip a coin: if it is heads (50% probability), we swap 2 job ids and quit.
  2. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 25% in total), we swap 3 job ids and quit.
  3. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 12.5% in total), we swap 4 job ids and quit.
  4. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 6.25% in total), we swap 5 job ids and quit.
  5. otherwise (it was tail), we again flip a coin. if it is heads (50% probability, now 3.125% in total), we swap 6 job ids and quit.
  6. and so on.
- We most often make small moves, but sometimes bigger ones.
- Thereotically, we could always escape from local any optima, but the probability may sometimes be very very small.

Create Modified Copy of Existing Solution 2

Listing: Swap a random number of sub-jobs.

```
public class JSSPUnaryOperatorNSwap
    implements IUnarySearchOperator<int[]> {

  public void apply(int[] x, int[] dest, Random random) {
    System.arraycopy(x, 0, dest, 0, x.length);
    int i = random.nextInt(dest.length);
    int first = dest[i];
    int last = first;

    boolean hasNext;
    do {
      hasNext = random.nextBoolean();
      inner: for (;;) {
        final int j = random.nextInt(dest.length);
        final int job_j = dest[j];
        if ((last != job_j) &&
            (hasNext || (first != job_j))) {
          dest[i] = job_j;
          i = j;
          last = job_j;
          break inner;
        }
      }
    } while (hasNext);
```

So what do we get?

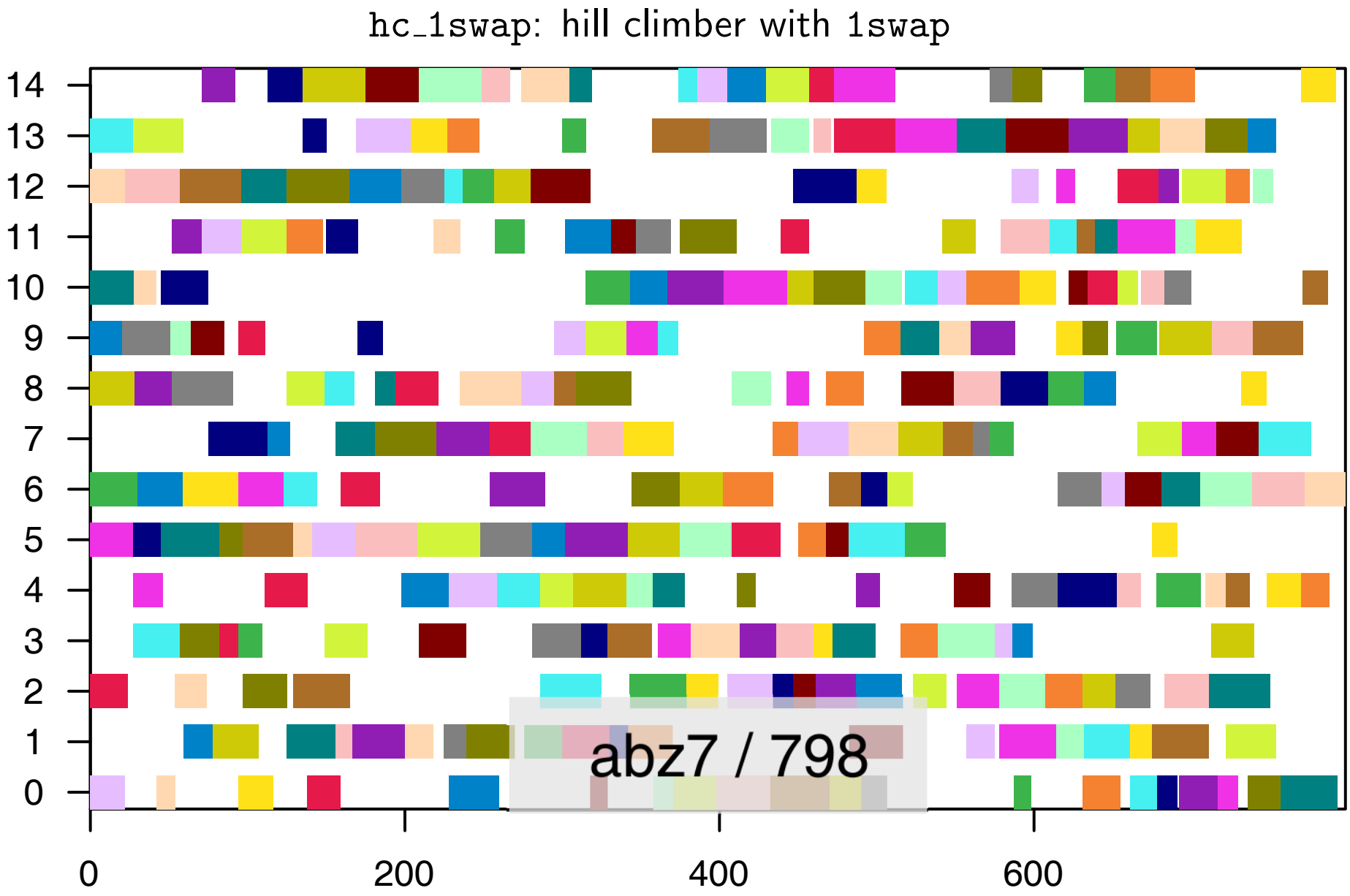
- I execute the program 101 times for each of the datasets `abz7`, `la24`, `swv15`, and `yn4`

So what do we get?

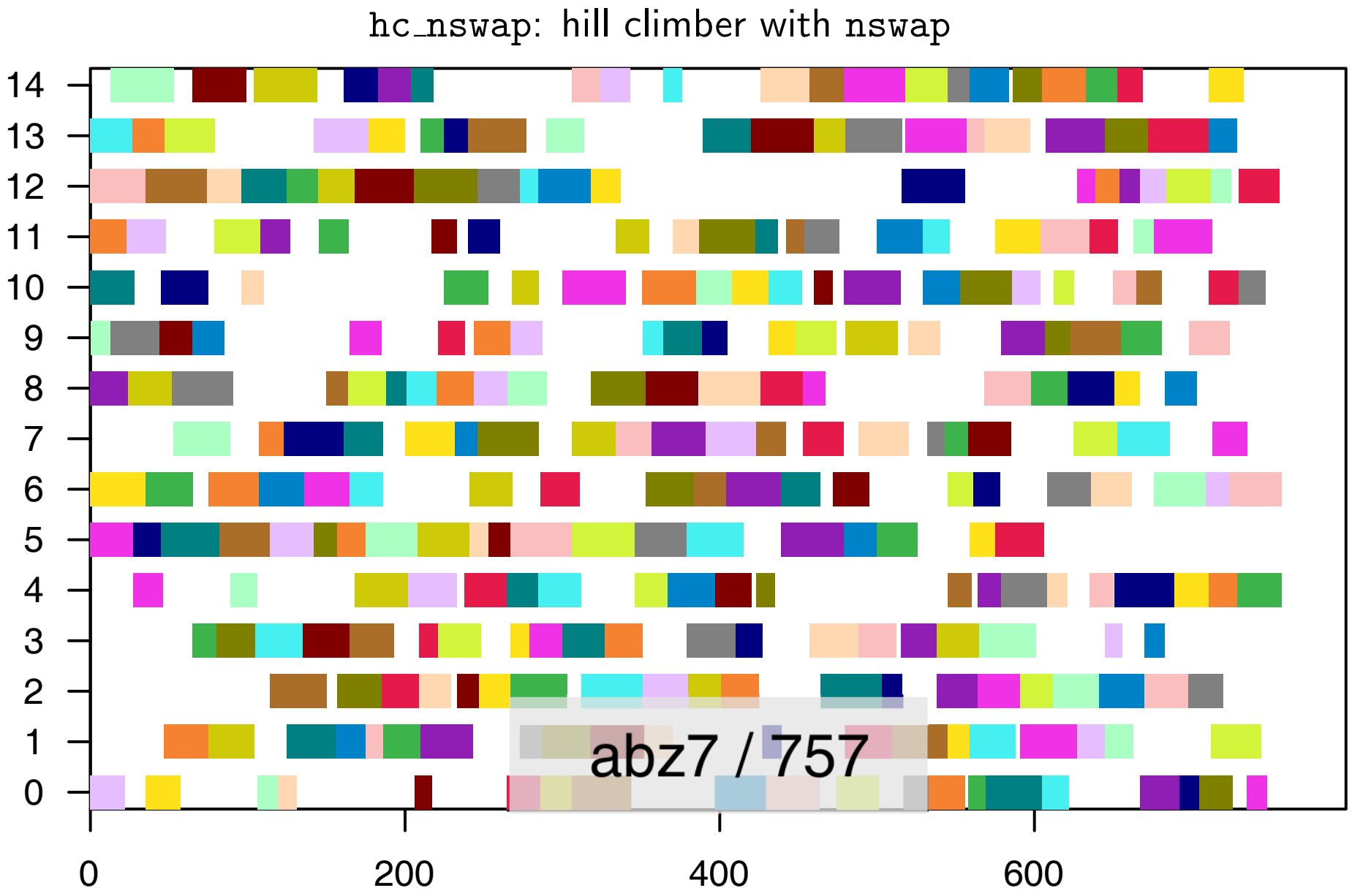
- I execute the program 101 times for each of the datasets `abz7`, `la24`, `swv15`, and `yn4`

		makespan				last improvement	
$\mathcal{I}$	algo	best	mean	med	sd	med(t)	med(FEs)
`abz7`	`hc_1swap`	**717**	800	798	28	0s	16'978
	`hc_nswap`	724	**757**	**757**	**17**	30s	8'145'596
`la24`	`hc_1swap`	999	1095	1086	56	0s	6612
	`hc_nswap`	**945**	**1017**	**1015**	**29**	21s	11'123'744
`swv15`	`hc_1swap`	3837	4108	4108	137	1s	104'598
	`hc_nswap`	**3599**	**3867**	**3859**	**113**	70s	11'559'667
`yn4`	`hc_1swap`	1109	1222	1220	48	0s	31'789
	`hc_nswap`	**1087**	**1160**	**1156**	**33**	63s	13'111'115

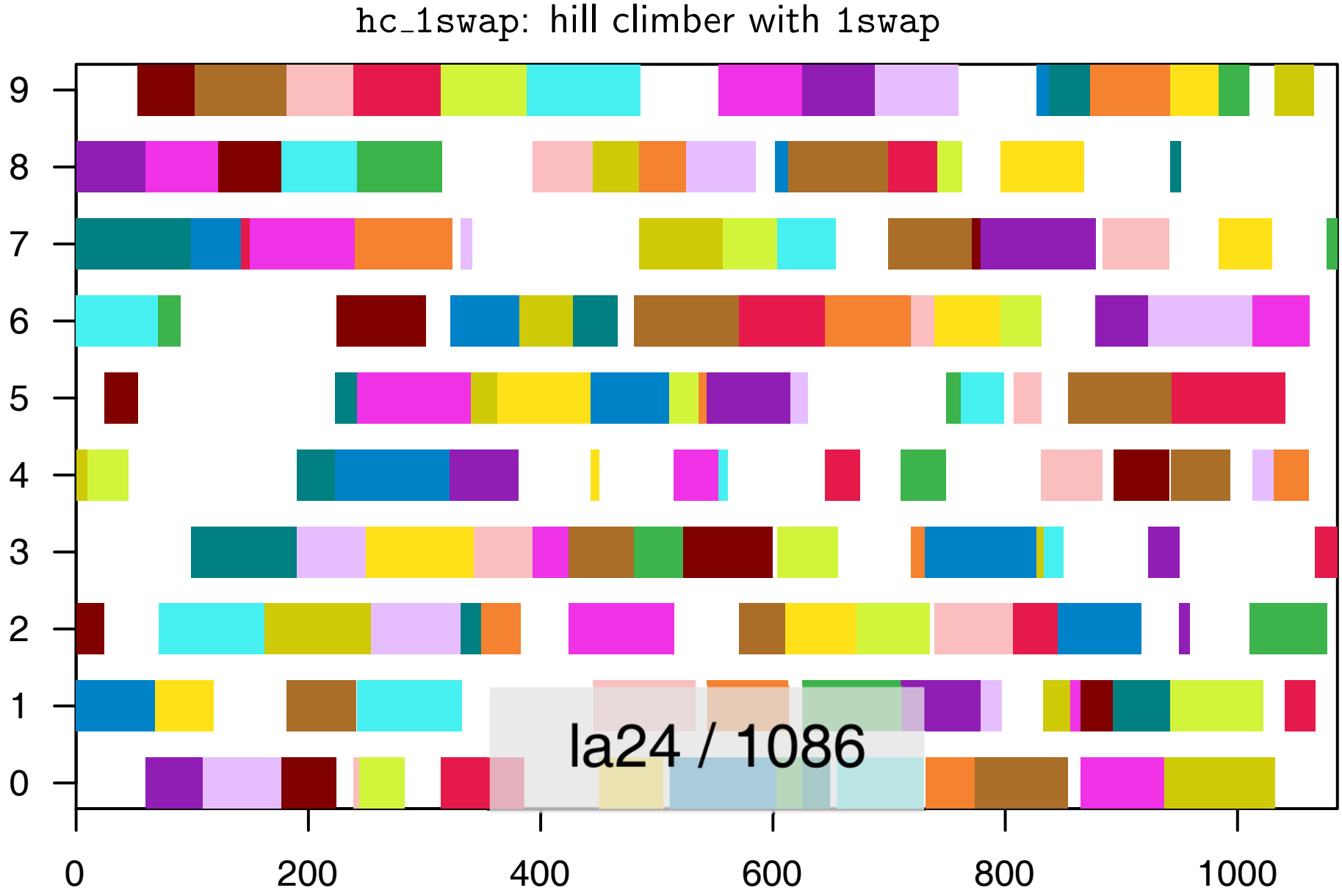
So what do we get?



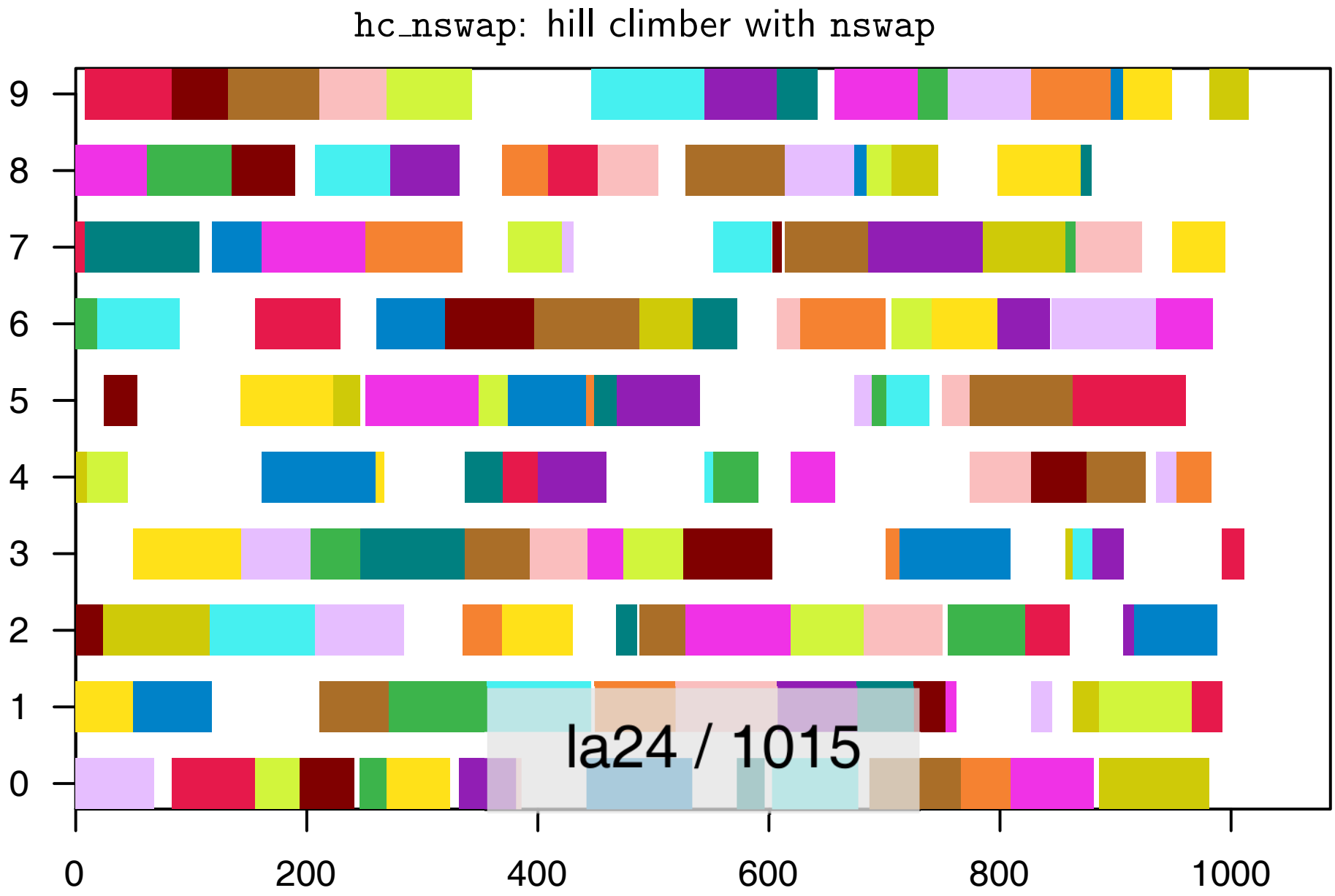
So what do we get?



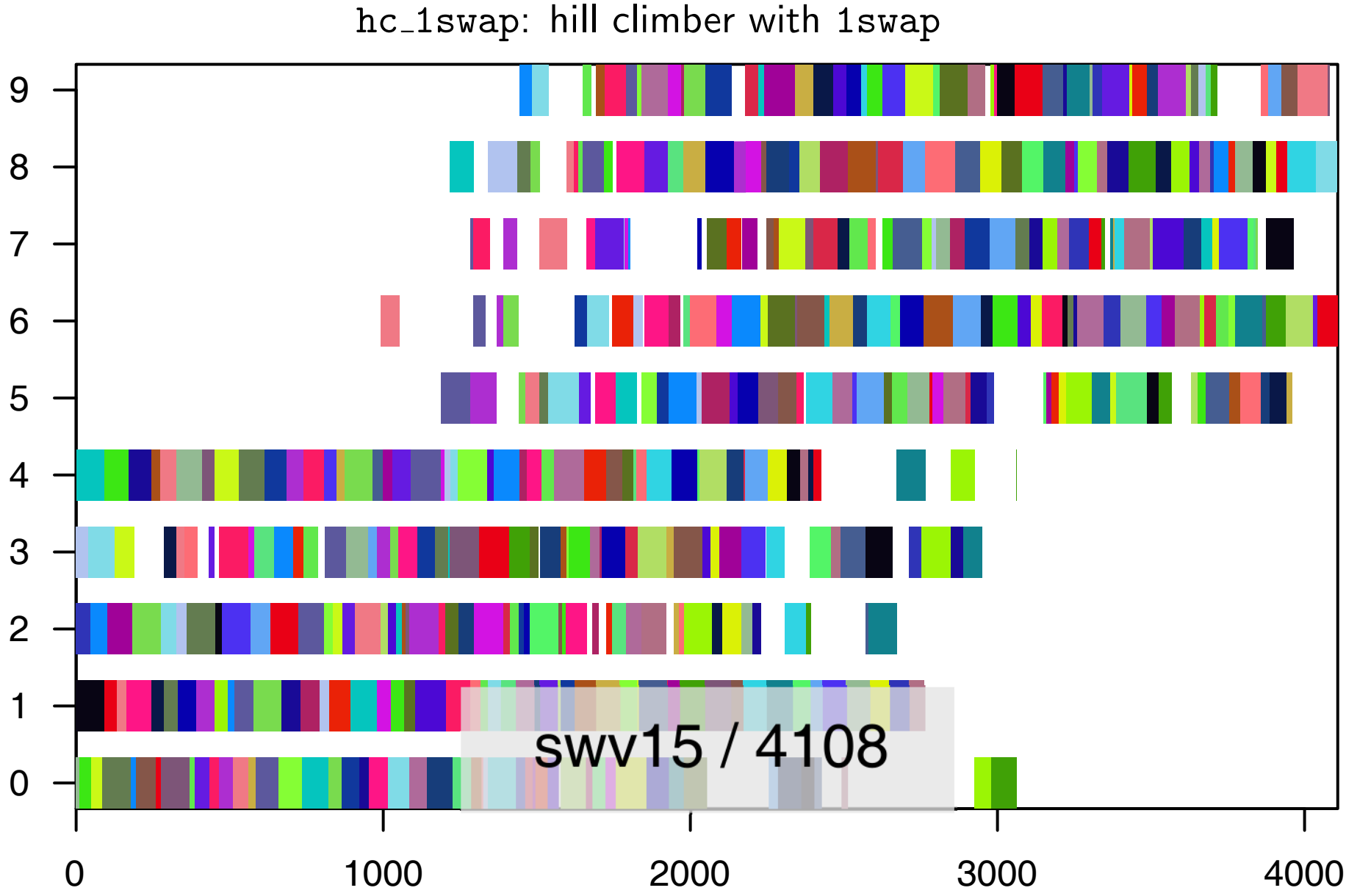
So what do we get?



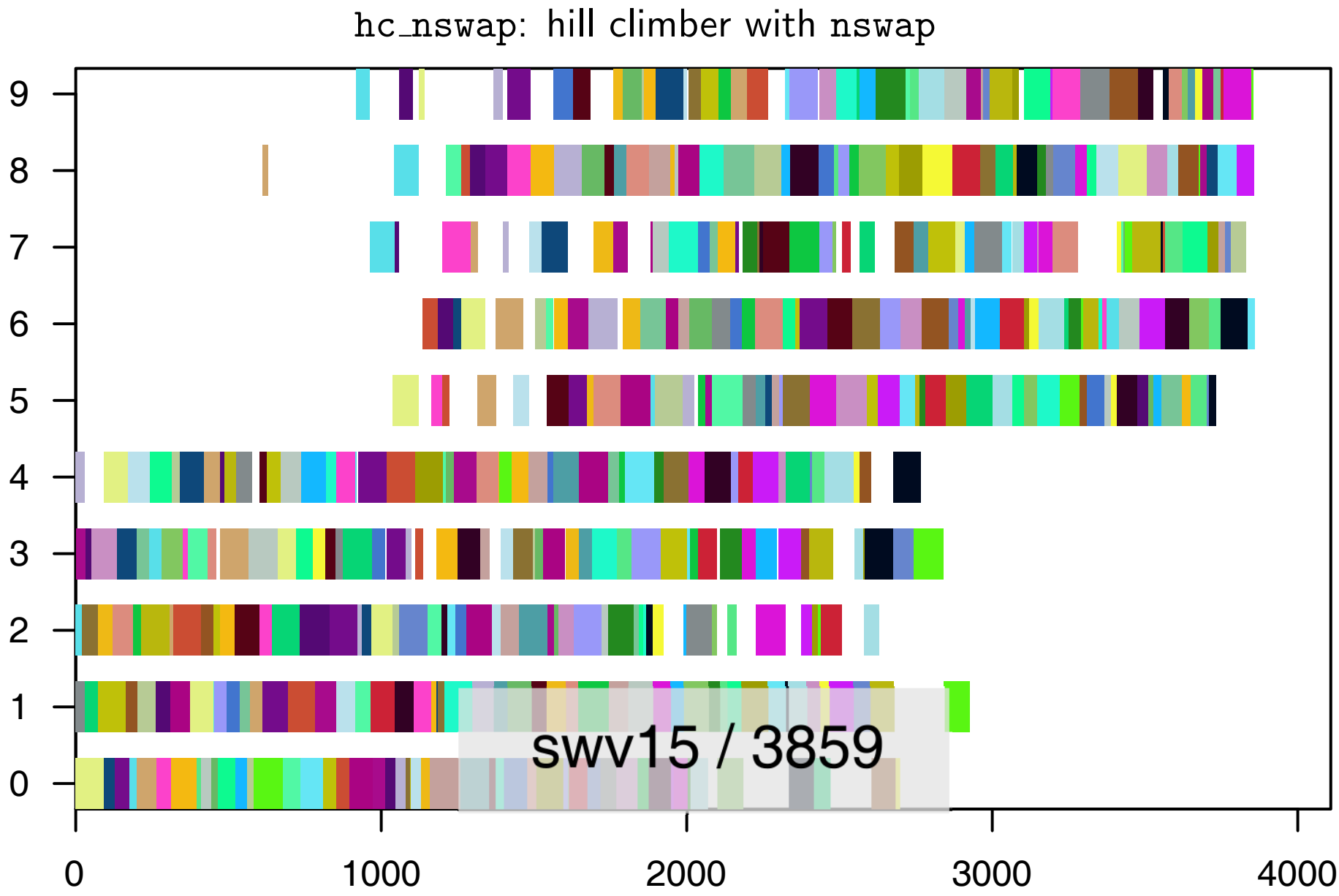
So what do we get?



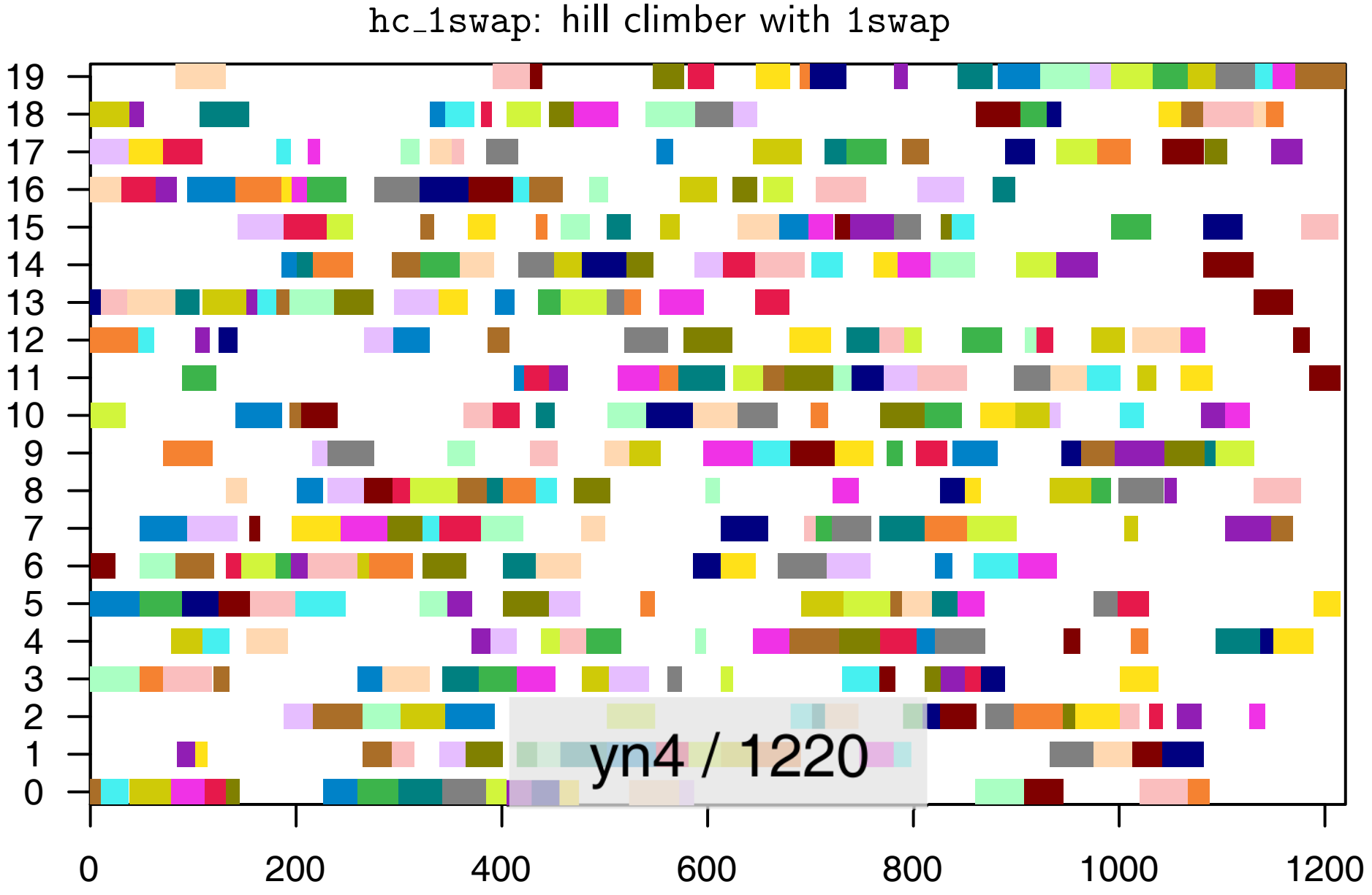
So what do we get?



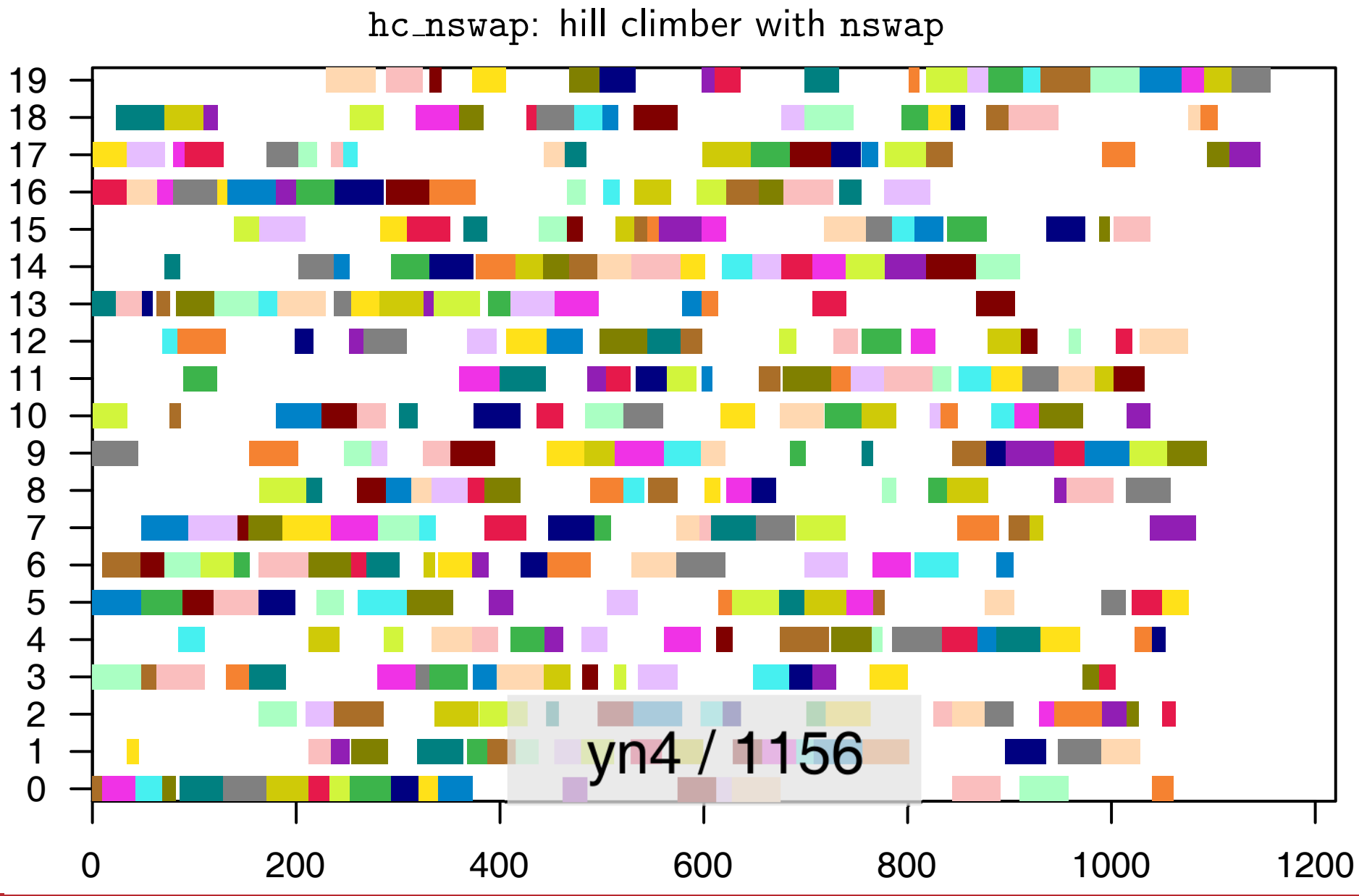
So what do we get?



So what do we get?



So what do we get?

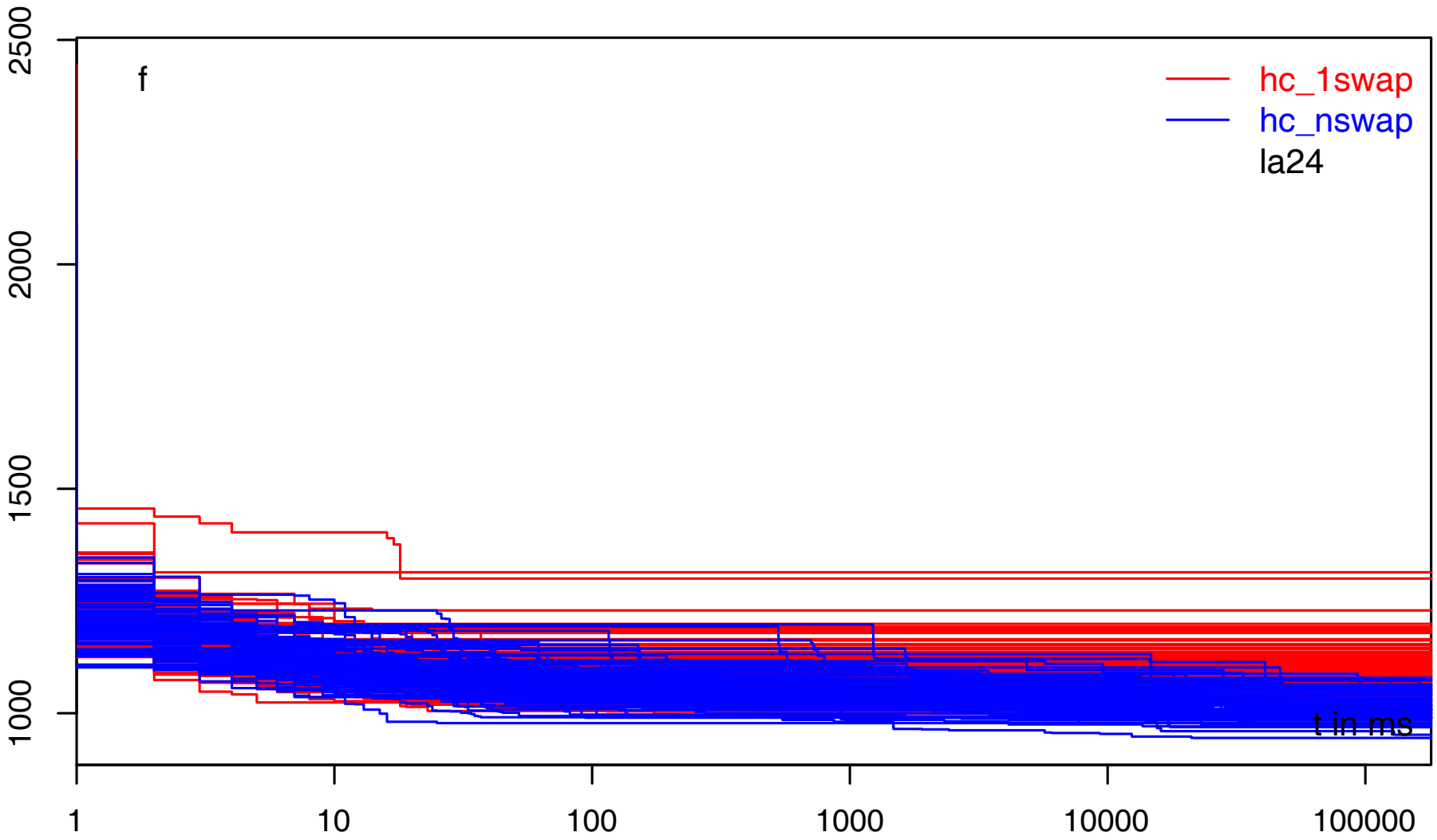


Progress over Time

What progress does the algorithm make over time?

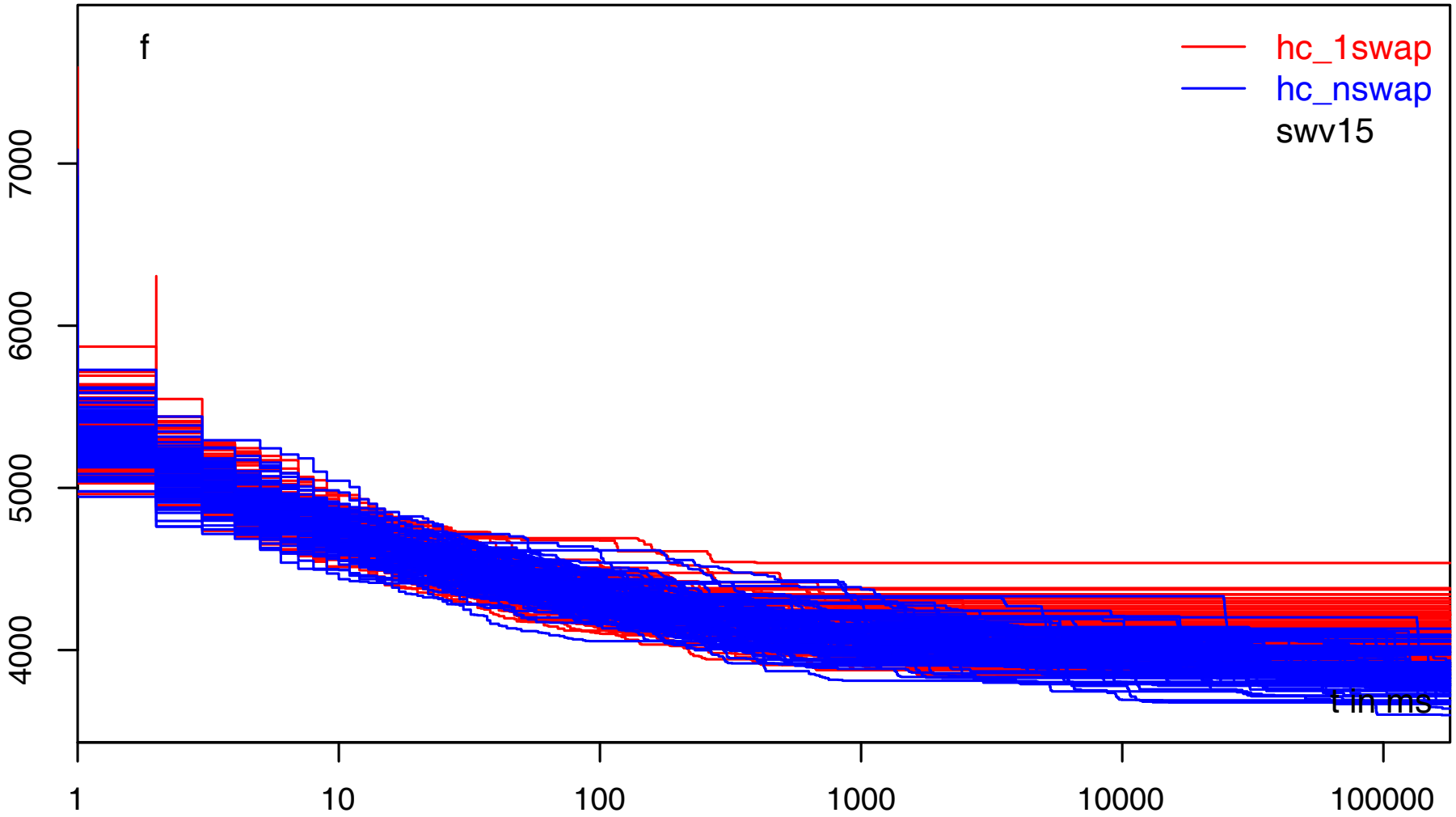
Progress over Time

What progress does the algorithm make over time?



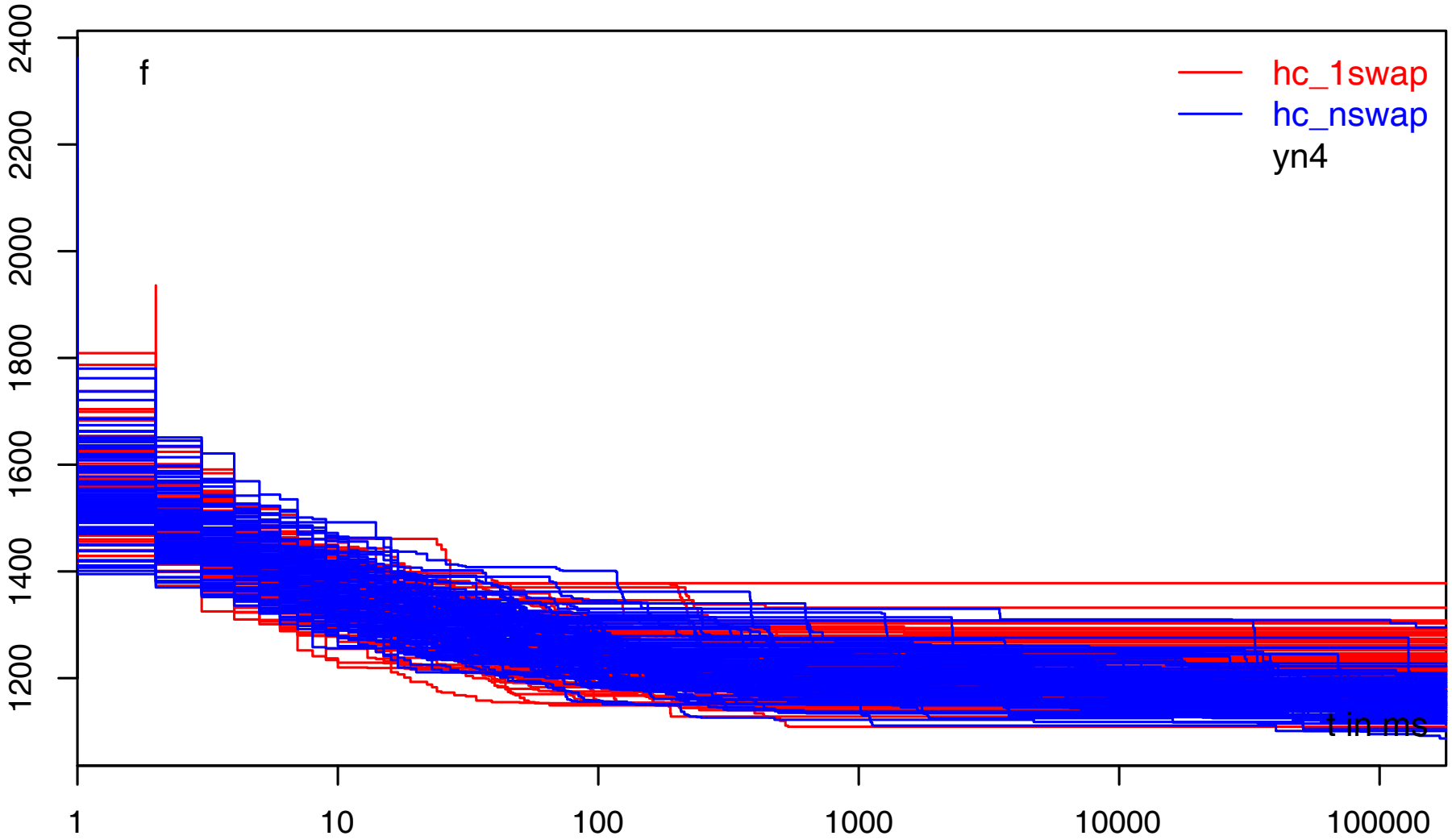
Progress over Time

What progress does the algorithm make over time?



Progress over Time

What progress does the algorithm make over time?



Section Outline

Combining the two ideas

- We had two entirely different ideas how to improve the hill climber.

Combining the two ideas

- We had two entirely different ideas how to improve the hill climber.
- Let's see how they work together!

So what do we get?

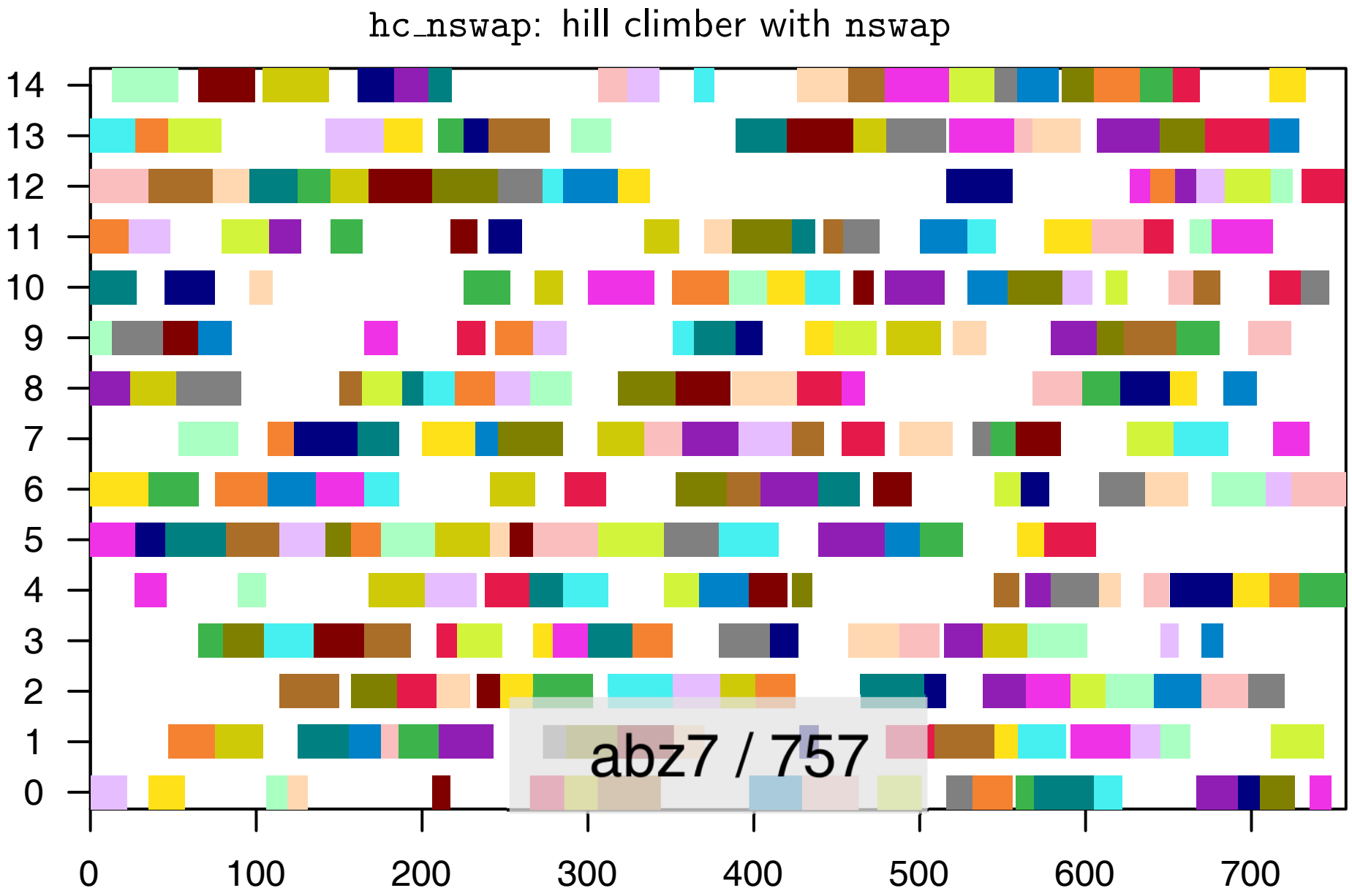
- I execute the program 101 times for each of the datasets `abz7`, `la24`, `swv15`, and `yn4`

So what do we get?

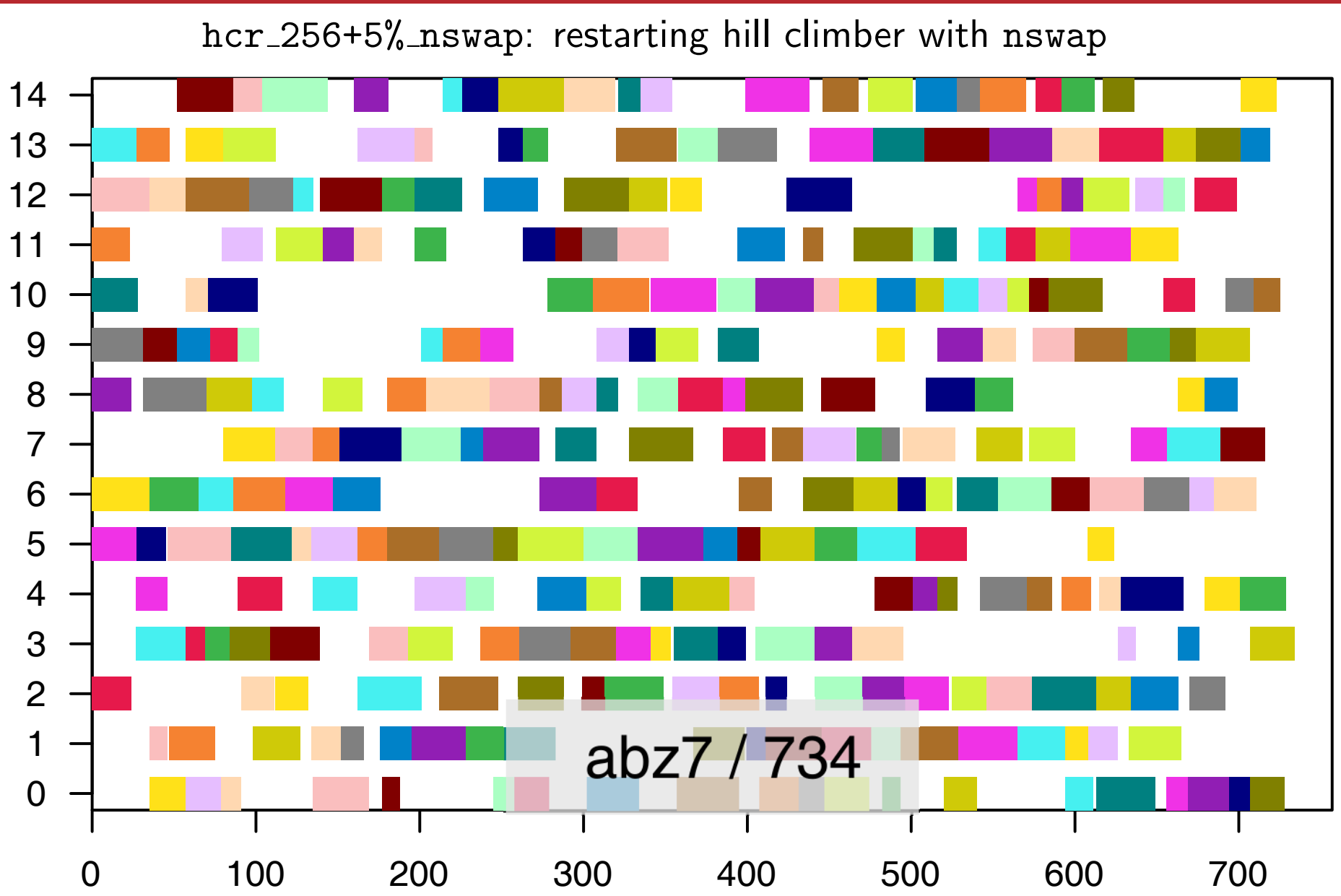
- I execute the program 101 times for each of the datasets `abz7`, `la24`, `swv15`, and `yn4`

		makespan				last improvement	
$\mathcal{I}$	algo	best	mean	med	sd	med(t)	med(FEs)
abz7	hc_1swap	717	800	798	28	0s	16978
	hcr_256+5%_1swap	723	742	743	**7**	21s	5681591
	hc_nswap	724	757	757	17	30s	8145596
	hcr_256+5%_nswap	**707**	**733**	**734**	7	64s	17293038
la24	hc_1swap	999	1095	1086	56	0s	6612
	hcr_256+5%_1swap	970	997	998	9	6s	3470368
	hc_nswap	**945**	1017	1015	29	21s	11123744
	hcr_256+5%_nswap	**945**	**981**	**984**	**9**	57s	29246097
swv15	hc_1swap	3837	4108	4108	137	1s	104598
	hcr_256+5%_1swap	3701	3850	3857	40	60s	9874102
	hc_nswap	**3599**	3867	3859	113	70s	11559667
	hcr_256+5%_nswap	3645	**3804**	**3811**	44	91s	14907737
yn4	hc_1swap	1109	1222	1220	48	0s	31789
	hcr_256+5%_1swap	1095	1129	1130	14	22s	4676669
	hc_nswap	1087	1160	1156	33	63s	13111115
	hcr_256+5%_nswap	**1081**	**1117**	**1119**	**14**	55s	11299461

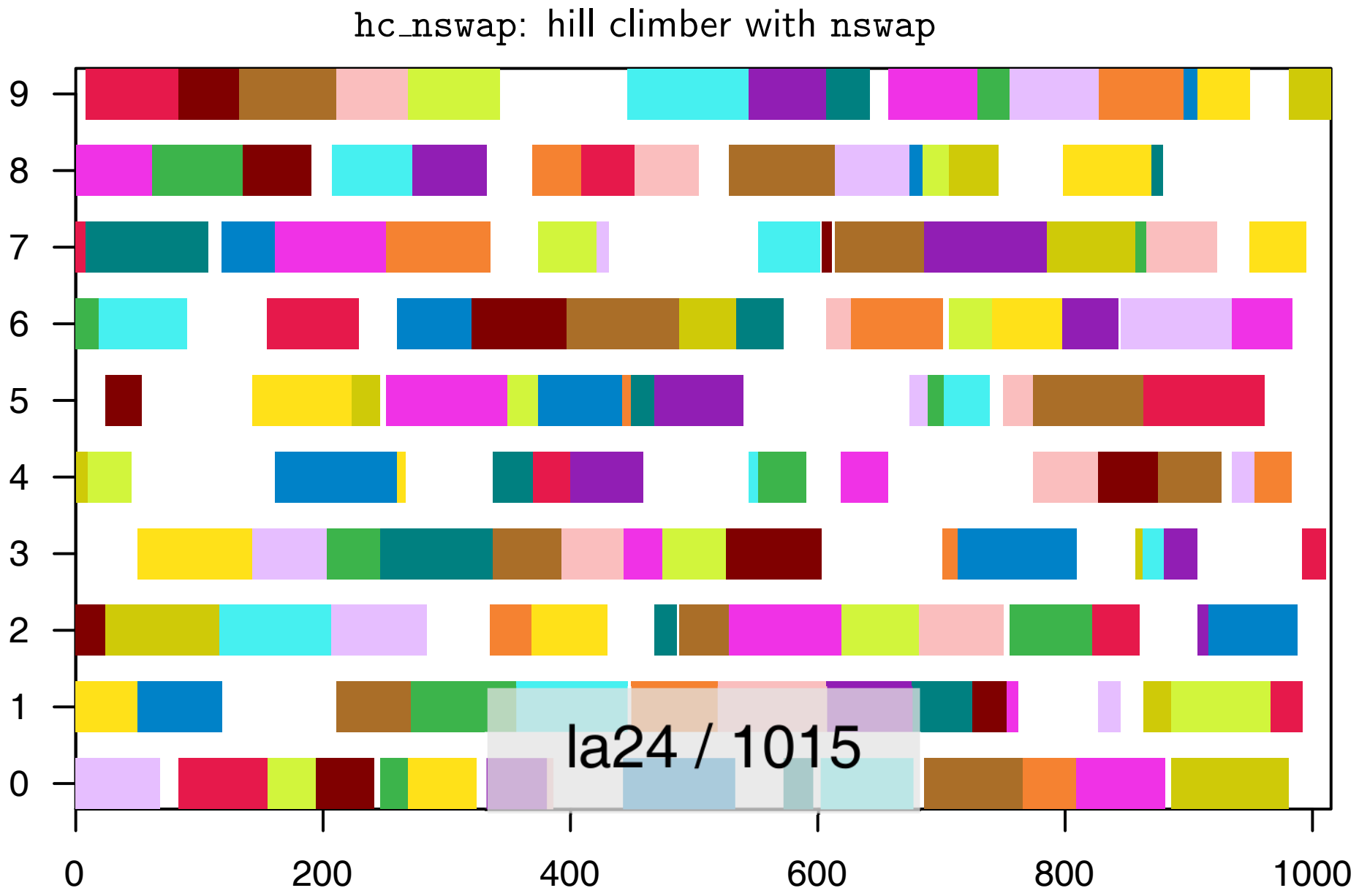
So what do we get?



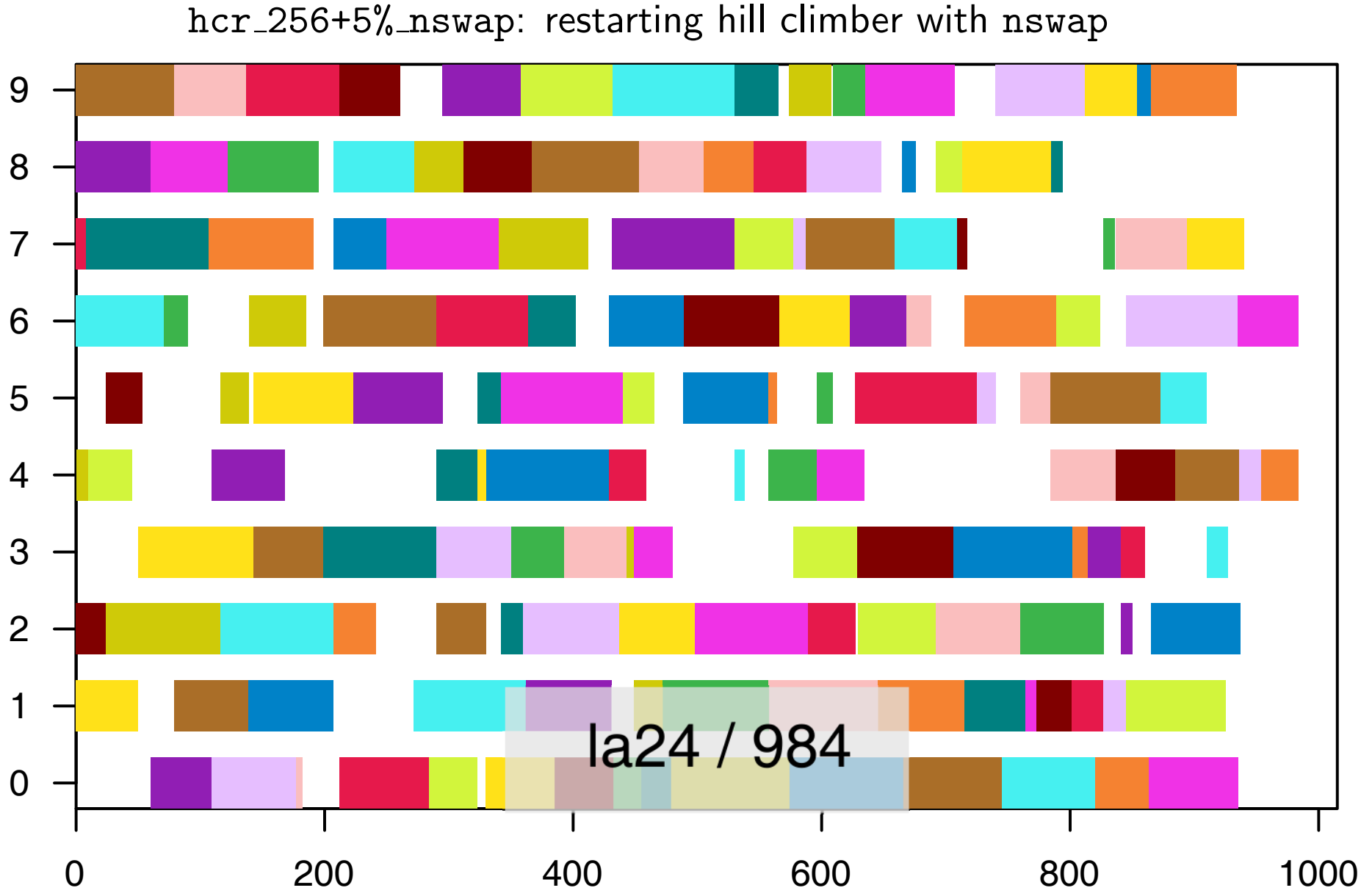
So what do we get?



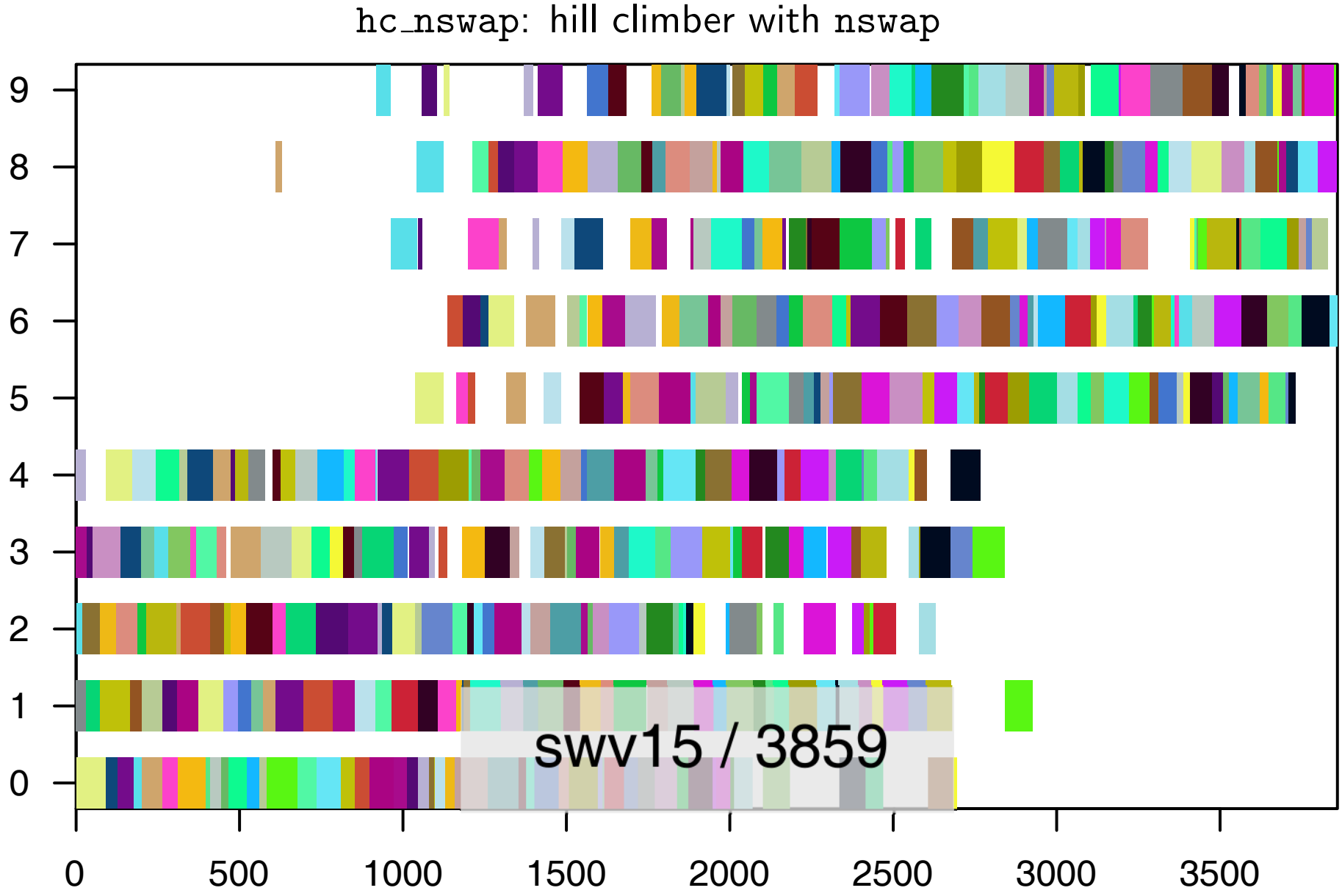
So what do we get?



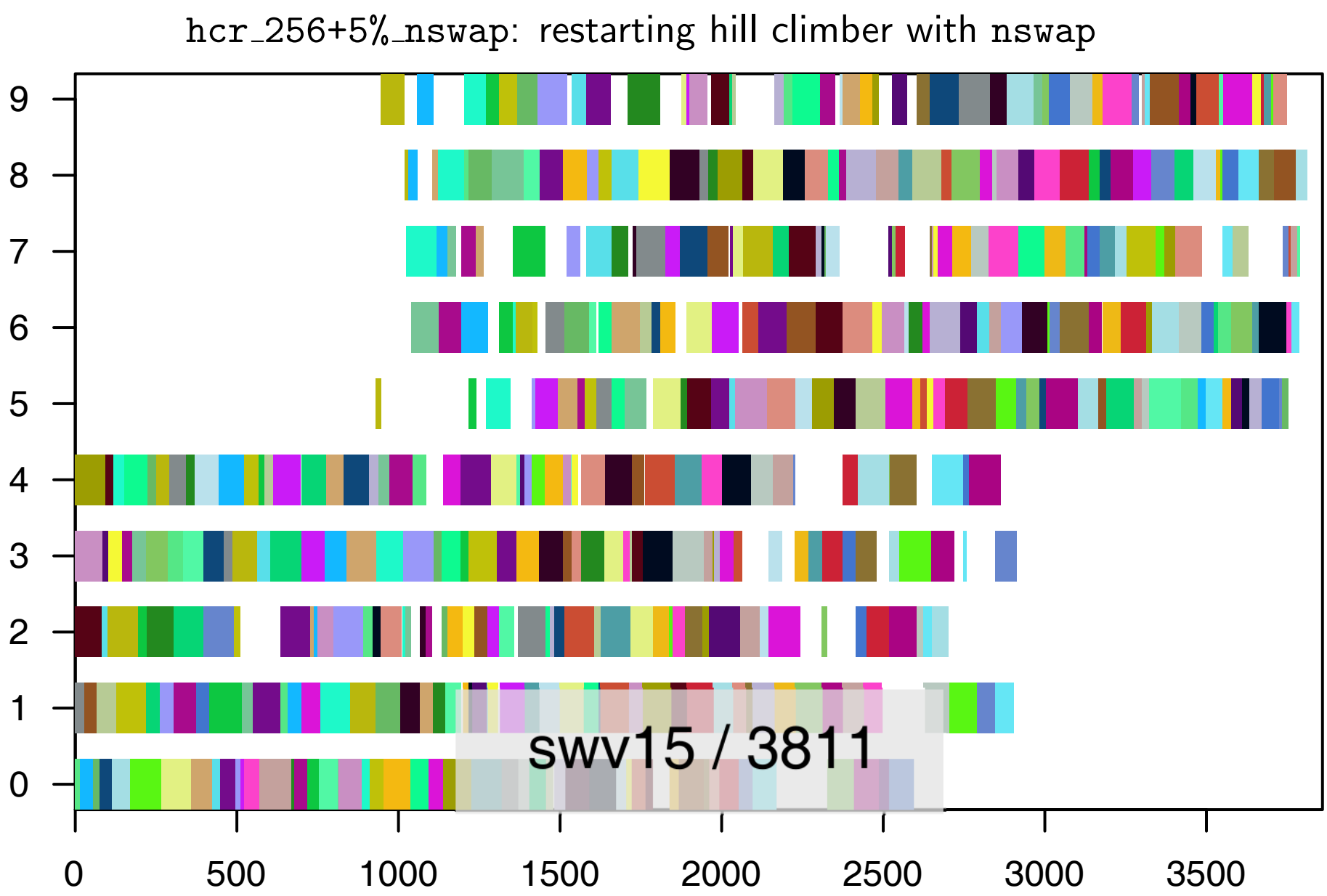
So what do we get?



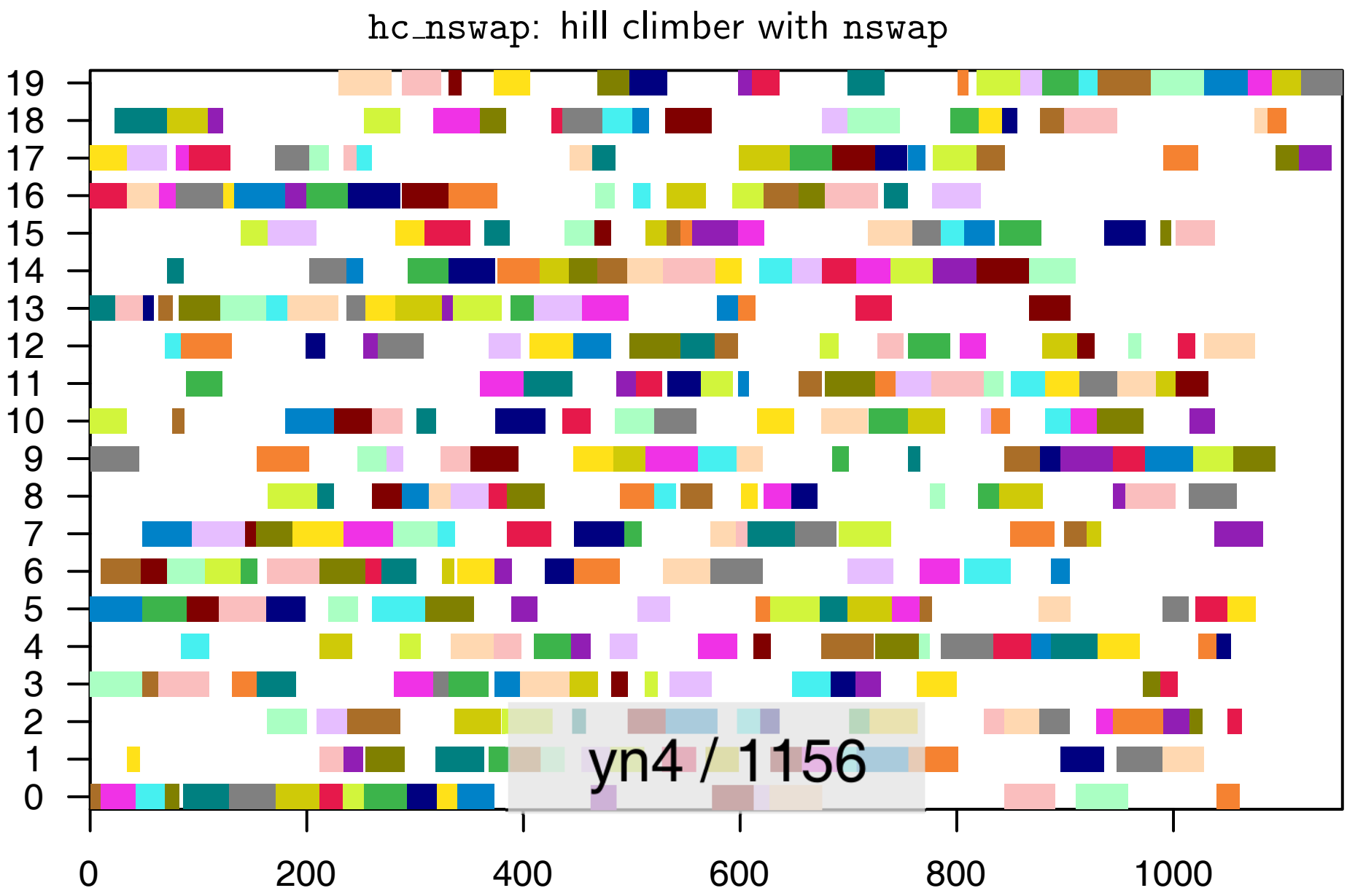
So what do we get?



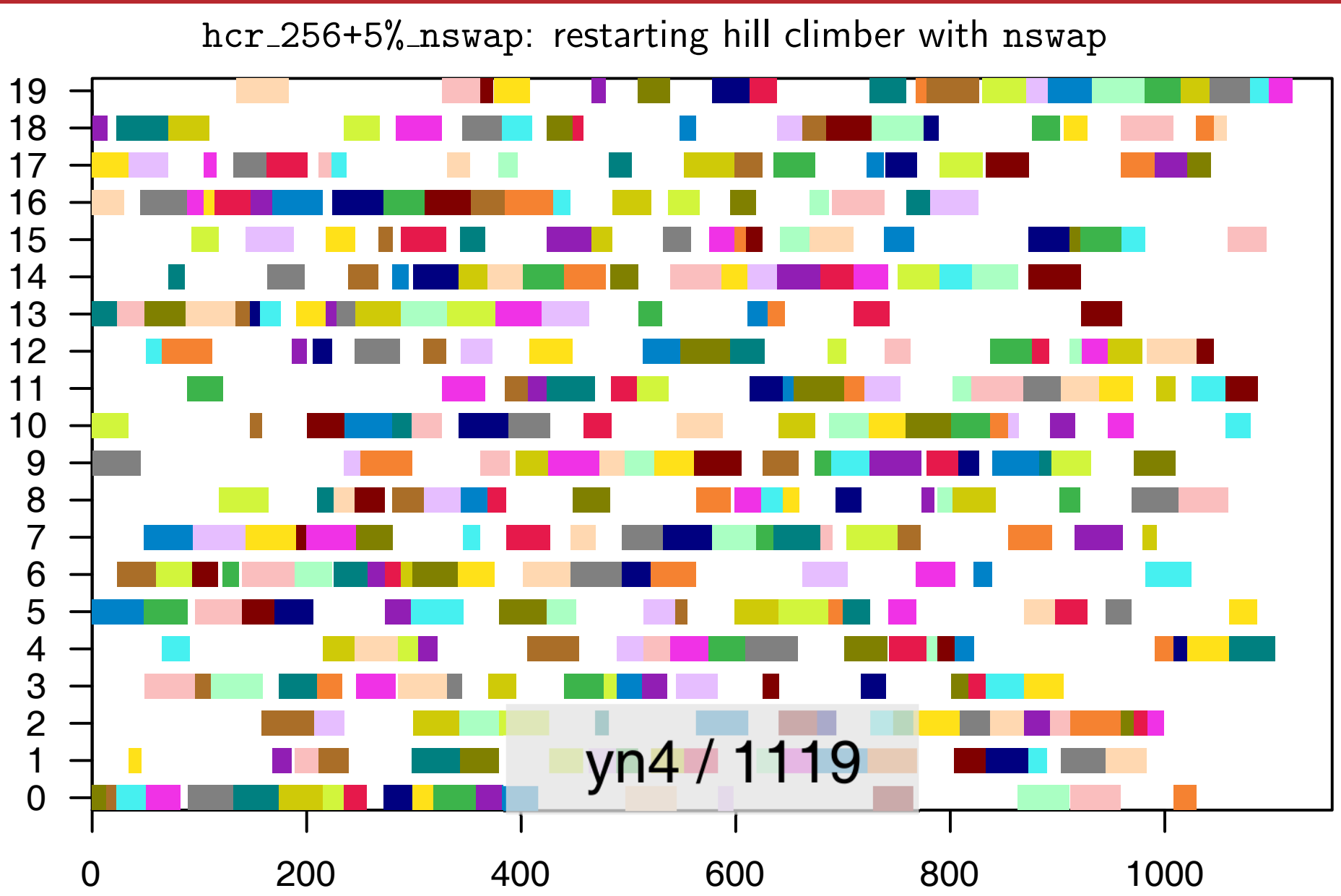
So what do we get?



So what do we get?



So what do we get?

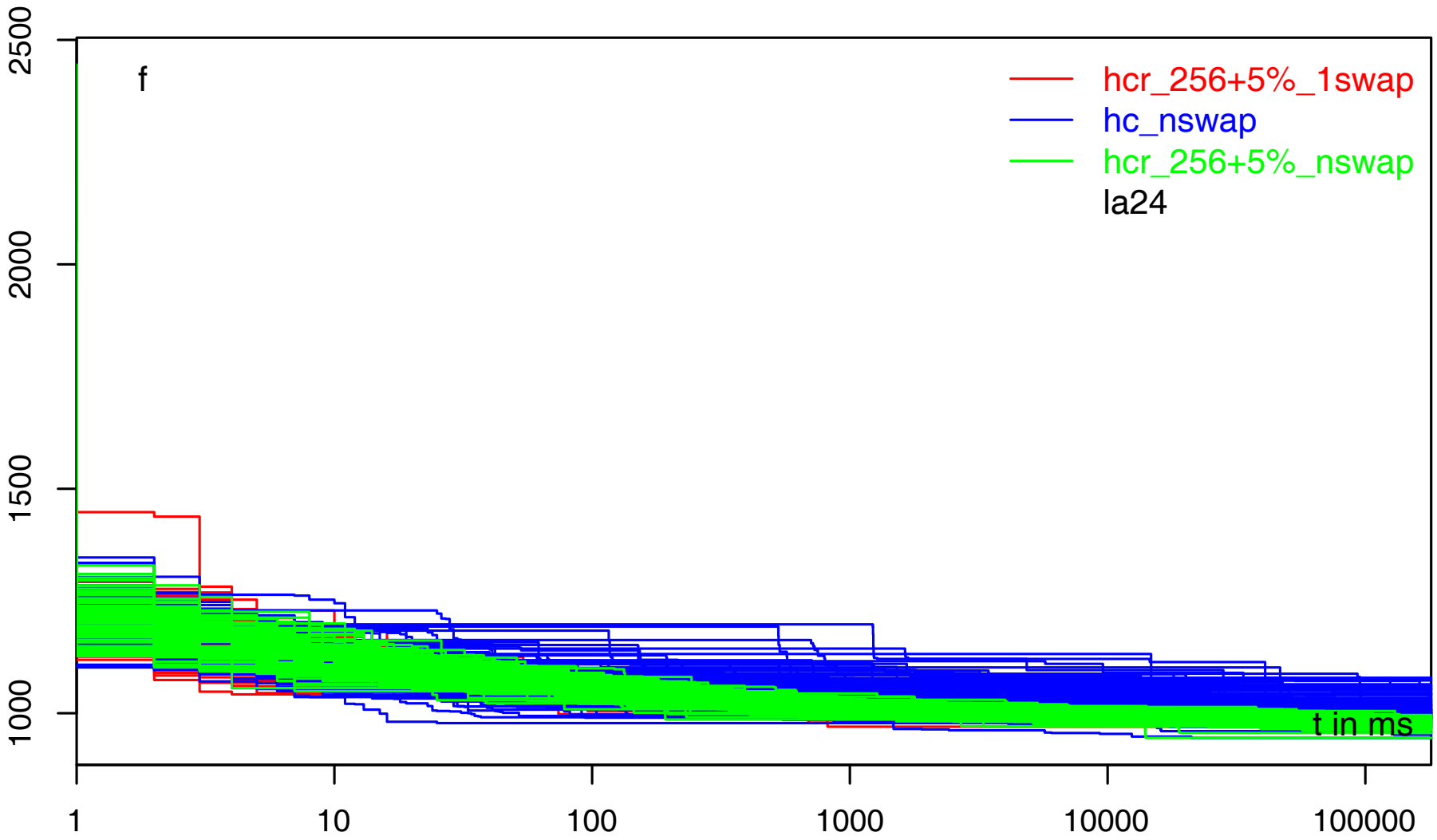


Progress over Time

What progress does the algorithm make over time?

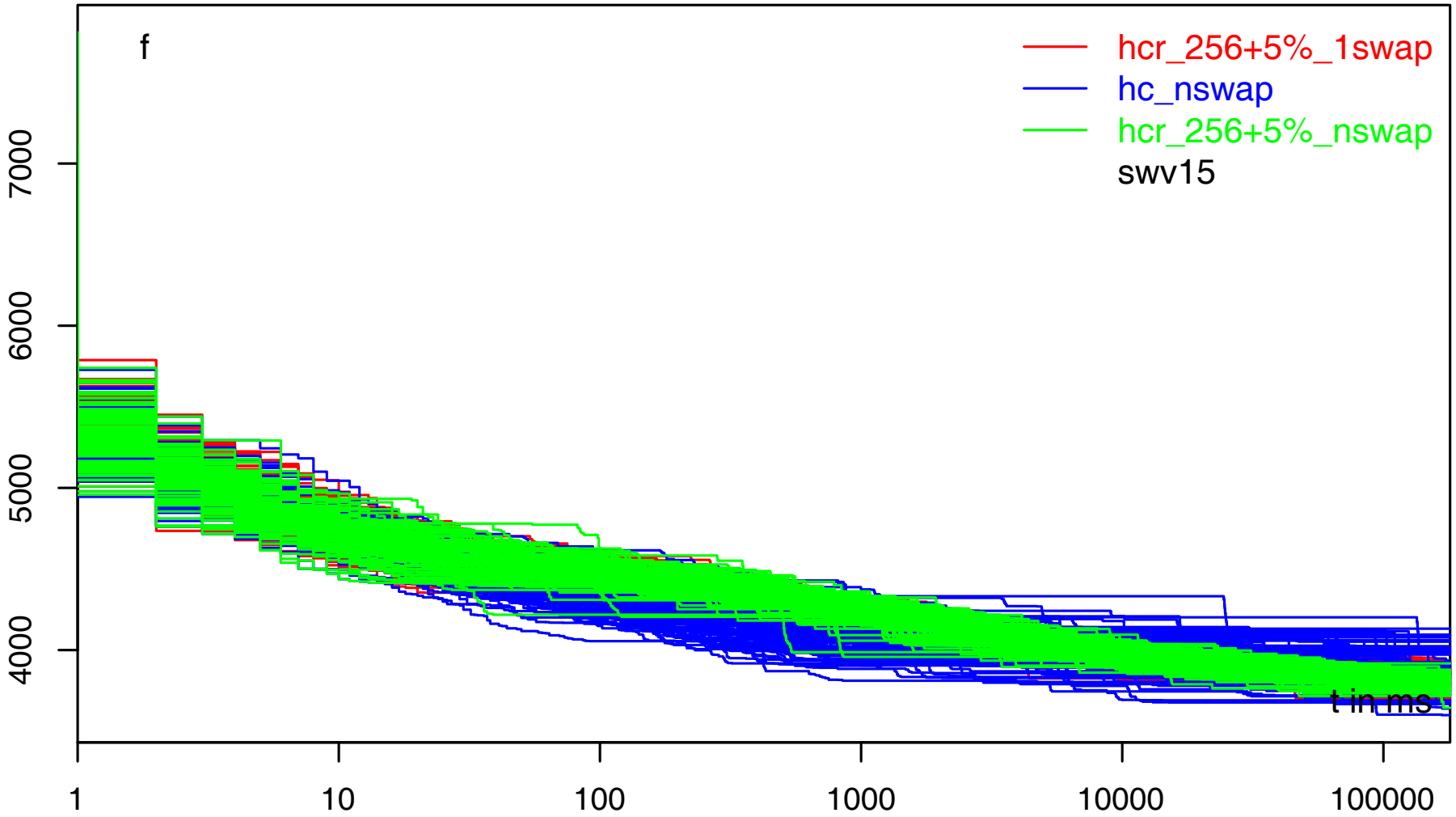
Progress over Time

What progress does the algorithm make over time?



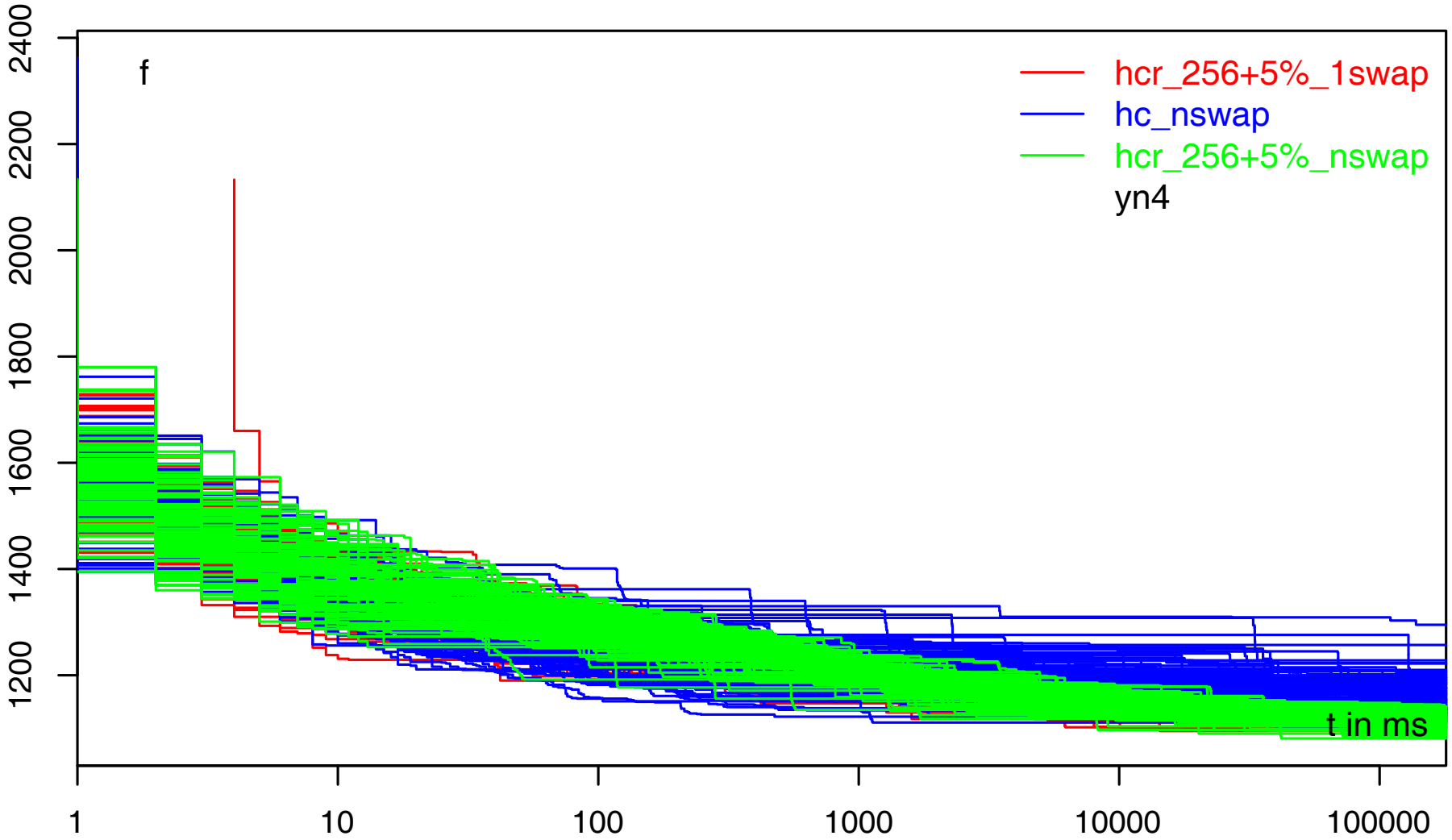
Progress over Time

What progress does the algorithm make over time?



Progress over Time

What progress does the algorithm make over time?



Summary

- By making use of the best point in the search space we have seen so far and iteratively trying to improve it, we can dramatically improve the results.

Summary

- By making use of the best point in the search space we have seen so far and iteratively trying to improve it, we can dramatically improve the results.
- We can further improve the results restarting the same algorithm from time to time if it has converged to a local optimum.

Summary

- By making use of the best point in the search space we have seen so far and iteratively trying to improve it, we can dramatically improve the results.
- We can further improve the results restarting the same algorithm from time to time if it has converged to a local optimum.
- We can also improve them by searching a nicer neighborhood.

Summary

- By making use of the best point in the search space we have seen so far and iteratively trying to improve it, we can dramatically improve the results.
- We can further improve the results restarting the same algorithm from time to time if it has converged to a local optimum.
- We can also improve them by searching a nicer neighborhood.
- And we can combine both concepts to get even better results.

Summary

- By making use of the best point in the search space we have seen so far and iteratively trying to improve it, we can dramatically improve the results.
- We can further improve the results restarting the same algorithm from time to time if it has converged to a local optimum.
- We can also improve them by searching a nicer neighborhood.
- And we can combine both concepts to get even better results.
- But we still sometimes get bad results.

Summary

- By making use of the best point in the search space we have seen so far and iteratively trying to improve it, we can dramatically improve the results.
- We can further improve the results restarting the same algorithm from time to time if it has converged to a local optimum.
- We can also improve them by searching a nicer neighborhood.
- And we can combine both concepts to get even better results.
- But we still sometimes get bad results.
- Because it is still the same algorithm.

Summary

- By making use of the best point in the search space we have seen so far and iteratively trying to improve it, we can dramatically improve the results.
- We can further improve the results restarting the same algorithm from time to time if it has converged to a local optimum.
- We can also improve them by searching a nicer neighborhood.
- And we can combine both concepts to get even better results.
- But we still sometimes get bad results.
- Because it is still the same algorithm.
- A hill climber can always get trapped in a local optimum, even with restarts. . . . . . if the basins of attraction of the local optima are larger than those of the global optimum.

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography

Bibliography I

1. Thomas Weise. *An Introduction to Optimization Algorithms*. Institute of Applied Optimization (IAO), Faculty of Computer Science and Technology, Hefei University, Hefei, Anhui, China, 2019-06-25 edition, 2018–2019. URL `http://thomasweise.github.io/aitoa/`. see also [2].
2. Thomas Weise. *Global Optimization Algorithms – Theory and Application*. it-weise.de (self-published), Germany, 2009. URL `http://www.it-weise.de/projects/book.pdf`.
3. Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier, 2005. ISBN 1493303732.
4. Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (AIMA)*. Prentice Hall International Inc., Upper Saddle River, NJ, USA, 2 edition, 2002. ISBN 0-13-080302-2.
5. James C. Spall. *Introduction to Stochastic Search and Optimization*, volume 6 of *Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimizationn*. Wiley Interscience, Chichester, West Sussex, UK, April 2003. ISBN 0-471-33052-3. URL `https://www.jhuapl.edu/ISSO/`.
6. Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Technische Universität Berlin, Berlin, Germany, 1971–1973.
7. Ingo Rechenberg. *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik*. Frommann-Holzboog Verlag, Bad Cannstadt, Stuttgart, Baden-Württemberg, Germany, 1994. ISBN 3-7728-1642-8.
8. Thomas Weise, Raymond Chiong, and Ke Tang. Evolutionary optimization: Pitfalls and booby traps. *Journal of Computer Science and Technology (JCST)*, 27:907–936, September 2012. doi: 10.1007/s11390-012-1274-4.
9. Thomas Weise, Michael Zapf, Raymond Chiong, and Antonio Jesús Nebro Urbaneja. Why is optimization difficult? In Raymond Chiong, editor, *Nature-Inspired Algorithms for Optimisation*, volume 193/2009 of *Studies in Computational Intelligence (SCI)*, chapter 1, pages 1–50. Springer-Verlag, Berlin/Heidelberg, April 2009. ISBN 978-3-642-00266-3. doi: 10.1007/978-3-642-00267-0_1.