合肥學院
HEFEI UNIVERSITY

Optimization Algorithms

7. Simulated Annealing

Thomas Weise · 汤卫思
tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn/5

Institute of Applied Optimization (IAO) | 应用优化研究所
School of Artificial Intelligence and Big Data | 人工智能与大数据学院
Hefei University | 合肥学院
Hefei, Anhui, China | 中国安徽省合肥市

Outline

1. Introduction
2. Algorithm Concept: Probabilistic Acceptance of Worse Solutions
3. Ingredient: Temperature Schedule
4. Algorithm Implementation
5. Configuring the Algorithm
6. Experiment and Analysis

Introduction

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search.

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.
- A pure hill climbing algorithm is likely to get stuck at local optima, which may vary in quality.

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.
- A pure hill climbing algorithm is likely to get stuck at local optima, which may vary in quality.
- We also found that we can utilize the variance of the result quality by restarting the optimization process when it could not improve any more.

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.
- A pure hill climbing algorithm is likely to get stuck at local optima, which may vary in quality.
- We also found that we can utilize the variance of the result quality by restarting the optimization process when it could not improve any more.
- But such a restart is costly, as it forces the local search to start completely from scratch (while we, of course, remember the best-ever solution in a variable hidden from the algorithm).

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.
- A pure hill climbing algorithm is likely to get stuck at local optima, which may vary in quality.
- We also found that we can utilize the variance of the result quality by restarting the optimization process when it could not improve any more.
- But such a restart is costly, as it forces the local search to start completely from scratch (while we, of course, remember the best-ever solution in a variable hidden from the algorithm).
- Alternatively, we also tried to use operators with larger neighborhoods.

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.
- A pure hill climbing algorithm is likely to get stuck at local optima, which may vary in quality.
- We also found that we can utilize the variance of the result quality by restarting the optimization process when it could not improve any more.
- But such a restart is costly, as it forces the local search to start completely from scratch (while we, of course, remember the best-ever solution in a variable hidden from the algorithm).
- Alternatively, we also tried to use operators with larger neighborhoods, but there will either still be local optima

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.
- A pure hill climbing algorithm is likely to get stuck at local optima, which may vary in quality.
- We also found that we can utilize the variance of the result quality by restarting the optimization process when it could not improve any more.
- But such a restart is costly, as it forces the local search to start completely from scratch (while we, of course, remember the best-ever solution in a variable hidden from the algorithm).
- Alternatively, we also tried to use operators with larger neighborhoods, but there will either still be local optima (if the neighborhood is not large enough)

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.
- A pure hill climbing algorithm is likely to get stuck at local optima, which may vary in quality.
- We also found that we can utilize the variance of the result quality by restarting the optimization process when it could not improve any more.
- But such a restart is costly, as it forces the local search to start completely from scratch (while we, of course, remember the best-ever solution in a variable hidden from the algorithm).
- Alternatively, we also tried to use operators with larger neighborhoods, but there will either still be local optima (if the neighborhood is not large enough) or there are points where it is hard to escape from

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.
- A pure hill climbing algorithm is likely to get stuck at local optima, which may vary in quality.
- We also found that we can utilize the variance of the result quality by restarting the optimization process when it could not improve any more.
- But such a restart is costly, as it forces the local search to start completely from scratch (while we, of course, remember the best-ever solution in a variable hidden from the algorithm).
- Alternatively, we also tried to use operators with larger neighborhoods, but there will either still be local optima (if the neighborhood is not large enough) or there are points where it is hard to escape from (if the neighborhood is very large but non-uniformly sampled, as our `nswap` operator does)

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.
- A pure hill climbing algorithm is likely to get stuck at local optima, which may vary in quality.
- We also found that we can utilize the variance of the result quality by restarting the optimization process when it could not improve any more.
- But such a restart is costly, as it forces the local search to start completely from scratch (while we, of course, remember the best-ever solution in a variable hidden from the algorithm).
- Alternatively, we also tried to use operators with larger neighborhoods, but there will either still be local optima (if the neighborhood is not large enough) or there are points where it is hard to escape from (if the neighborhood is very large but non-uniformly sampled, as our `nswap` operator does) or the search will get very slow

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.
- A pure hill climbing algorithm is likely to get stuck at local optima, which may vary in quality.
- We also found that we can utilize the variance of the result quality by restarting the optimization process when it could not improve any more.
- But such a restart is costly, as it forces the local search to start completely from scratch (while we, of course, remember the best-ever solution in a variable hidden from the algorithm).
- Alternatively, we also tried to use operators with larger neighborhoods, but there will either still be local optima (if the neighborhood is not large enough) or there are points where it is hard to escape from (if the neighborhood is very large but non-uniformly sampled, as our `nswap` operator does) or the search will get very slow (if the neighborhood is very large and uniformly sampled).

Local Search and Hill Climbing

- So far, we have only discussed one variant of local search: the stochastic hill climbing algorithm.
- A pure hill climbing algorithm is likely to get stuck at local optima, which may vary in quality.
- We also found that we can utilize the variance of the result quality by restarting the optimization process when it could not improve any more.
- But such a restart is costly, as it forces the local search to start completely from scratch (while we, of course, remember the best-ever solution in a variable hidden from the algorithm).
- Alternatively, we also tried to use operators with larger neighborhoods, but there will either still be local optima (if the neighborhood is not large enough) or there are points where it is hard to escape from (if the neighborhood is very large but non-uniformly sampled, as our `nswap` operator does) or the search will get very slow (if the neighborhood is very large and uniformly sampled).
- So, for now, let's stick with the `1swap` operator.

Idea

- A schedule which is a local optimum (under `1swap`) probably is at least somewhat similar to what the globally optimal schedule would look like.

Idea

- A schedule which is a local optimum (under `1swap`) probably is at least somewhat similar to what the globally optimal schedule would look like.
- It must, obviously, also be somewhat different (otherwise it would be the global optimum already).

Idea

- A schedule which is a local optimum (under `1swap`) probably is at least somewhat similar to what the globally optimal schedule would look like.
- It must, obviously, also be somewhat different (otherwise it would be the global optimum already).
- This difference is shaped such that it cannot be conquered by the `1swap` unary search operator that we use.

Idea

- A schedule which is a local optimum (under `1swap`) probably is at least somewhat similar to what the globally optimal schedule would look like.
- It must, obviously, also be somewhat different (otherwise it would be the global optimum already).
- This difference is shaped such that it cannot be conquered by the `1swap` unary search operator that we use.
- If we do a restart, we also dispose of the similarities to the global optimum that we have already discovered.

Idea

- A schedule which is a local optimum (under `1swap`) probably is at least somewhat similar to what the globally optimal schedule would look like.
- It must, obviously, also be somewhat different (otherwise it would be the global optimum already).
- This difference is shaped such that it cannot be conquered by the `1swap` unary search operator that we use.
- If we do a restart, we also dispose of the similarities to the global optimum that we have already discovered.
- Then, we will subsequently spend time to re-discover them in the hope that this will happen in a way that allows us to eventually reach the global optimum itself (or at least a better local optimum).

Idea

- A schedule which is a local optimum (under `1swap`) probably is at least somewhat similar to what the globally optimal schedule would look like.
- It must, obviously, also be somewhat different (otherwise it would be the global optimum already).
- This difference is shaped such that it cannot be conquered by the `1swap` unary search operator that we use.
- If we do a restart, we also dispose of the similarities to the global optimum that we have already discovered.
- Then, we will subsequently spend time to re-discover them in the hope that this will happen in a way that allows us to eventually reach the global optimum itself (or at least a better local optimum).
- Can there be a less-costly way?

Algorithm Concept: Probabilistic Acceptance of Worse Solutions

Simulated Annealing

- Simulated Annealing (SA)[3–6] is a local search which provides another approach to escape from local optima[2 7].

Simulated Annealing

- Simulated Annealing (SA)[3–6] is a local search which provides another approach to escape from local optima[2 7].
- Instead of restarting the algorithm when reaching a local optima, it tries to preserve the parts of the current solution by sometimes permitting search steps towards worsening objective values.

Simulated Annealing

- Simulated Annealing (SA)[3–6] is a local search which provides another approach to escape from local optima[2 7].
- Instead of restarting the algorithm when reaching a local optima, it tries to preserve the parts of the current solution by sometimes permitting search steps towards worsening objective values.
- This algorithm therefore introduces three principles

Simulated Annealing

- Simulated Annealing (SA)[3–6] is a local search which provides another approach to escape from local optima[2,7].
- Instead of restarting the algorithm when reaching a local optima, it tries to preserve the parts of the current solution by sometimes permitting search steps towards worsening objective values.
- This algorithm therefore introduces three principles:
    1. Worse candidate solutions are sometimes accepted, too.

Simulated Annealing

- Simulated Annealing (SA)[3–6] is a local search which provides another approach to escape from local optima[2, 7].
- Instead of restarting the algorithm when reaching a local optima, it tries to preserve the parts of the current solution by sometimes permitting search steps towards worsening objective values.
- This algorithm therefore introduces three principles:
  1. Worse candidate solutions are sometimes accepted, too.
  2. The probability P of accepting them is decreases with increasing differences ΔE of the objective values to the current solution.

Simulated Annealing

- Simulated Annealing (SA)[3–6] is a local search which provides another approach to escape from local optima[2 7].
- Instead of restarting the algorithm when reaching a local optima, it tries to preserve the parts of the current solution by sometimes permitting search steps towards worsening objective values.
- This algorithm therefore introduces three principles:
    1. Worse candidate solutions are sometimes accepted, too.
    2. The probability P of accepting them is decreases with increasing differences ΔE of the objective values to the current solution.
    3. The probability also decreases with the number of performed search steps.

Simulated Annealing

- Simulated Annealing (SA)[3–6] is a local search which provides another approach to escape from local optima[2 7].
- Instead of restarting the algorithm when reaching a local optima, it tries to preserve the parts of the current solution by sometimes permitting search steps towards worsening objective values.
- This algorithm therefore introduces three principles:
  1. Worse candidate solutions are sometimes accepted, too.
  2. The probability P of accepting them is decreases with increasing differences ΔE of the objective values to the current solution.
  3. The probability also decreases with the number of performed search steps.
- These principles are "injected" into the basic loop of the hill climber.

Simulated Annealing

- Simulated Annealing (SA)[3–6] is a local search which provides another approach to escape from local optima[2 7].
- Instead of restarting the algorithm when reaching a local optima, it tries to preserve the parts of the current solution by sometimes permitting search steps towards worsening objective values.
- This algorithm therefore introduces three principles:
    1. Worse candidate solutions are sometimes accepted, too.
    2. The probability P of accepting them is decreases with increasing differences ΔE of the objective values to the current solution.
    3. The probability also decreases with the number of performed search steps.
- These principles are "injected" into the basic loop of the hill climber.
- How can we implement these concepts?

Acceptance Probability

- How can we implement these concepts?

Acceptance Probability

- How can we implement these concepts?
- Let's assume that the "current" point in the search space known by our local search is $x \in \mathbb{X}$.

Acceptance Probability

- How can we implement these concepts?
- Let's assume that the "current" point in the search space known by our local search is $x \in \mathbb{X}$ and that we have derived a new point $x' \in \mathbb{X}$ from it.

Acceptance Probability

- How can we implement these concepts?
- Let's assume that the "current" point in the search space known by our local search is $x \in \mathbb{X}$ and that we have derived a new point $x' \in \mathbb{X}$ from it using the unary search operator.

Acceptance Probability

- How can we implement these concepts?
- Let's assume that the "current" point in the search space known by our local search is $x \in \mathbb{X}$ and that we have derived a new point $x' \in \mathbb{X}$ from it using the unary search operator.
- ΔE then be the difference between the objective value of x' and x.

Acceptance Probability

- How can we implement these concepts?
- Let's assume that the "current" point in the search space known by our local search is $x \in \mathbb{X}$ and that we have derived a new point $x' \in \mathbb{X}$ from it using the unary search operator.
- ΔE then be the difference between the objective value of x' and x:

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

Acceptance Probability

- How can we implement these concepts?
- Let's assume that the "current" point in the search space known by our local search is $x \in \mathbb{X}$ and that we have derived a new point $x' \in \mathbb{X}$ from it using the unary search operator.
- ΔE then be the difference between the objective value of x' and x:

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- $\Delta E < 0$ means that?

Acceptance Probability

- How can we implement these concepts?
- Let's assume that the "current" point in the search space known by our local search is $x \in \mathbb{X}$ and that we have derived a new point $x' \in \mathbb{X}$ from it using the unary search operator.
- ΔE then be the difference between the objective value of x' and x:

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- $\Delta E < 0$ means that the new x' is better than x since $f(\gamma(x')) < f(\gamma(x))$.

Acceptance Probability

- How can we implement these concepts?
- Let's assume that the "current" point in the search space known by our local search is $x \in \mathbb{X}$ and that we have derived a new point $x' \in \mathbb{X}$ from it using the unary search operator.
- ΔE then be the difference between the objective value of x' and x:

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- $\Delta E < 0$ means that the new x' is better than x since $f(\gamma(x')) < f(\gamma(x))$.
- $\Delta E > 0$ means that?

Acceptance Probability

- How can we implement these concepts?
- Let's assume that the "current" point in the search space known by our local search is $x \in \mathbb{X}$ and that we have derived a new point $x' \in \mathbb{X}$ from it using the unary search operator.
- ΔE then be the difference between the objective value of x' and x:

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- $\Delta E < 0$ means that the new x' is better than x since $f(\gamma(x')) < f(\gamma(x))$.
- $\Delta E > 0$ means that the new solution is worse.

Acceptance Probability

- How can we implement these concepts?
- Let's assume that the "current" point in the search space known by our local search is $x \in \mathbb{X}$ and that we have derived a new point $x' \in \mathbb{X}$ from it using the unary search operator.
- ΔE then be the difference between the objective value of x' and x:

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- $\Delta E < 0$ means that the new x' is better than x since $f(\gamma(x')) < f(\gamma(x))$.
- $\Delta E > 0$ means that the new solution is worse.
- $\Delta E = 0$ means that?

Acceptance Probability

- How can we implement these concepts?
- Let's assume that the "current" point in the search space known by our local search is $x \in \mathbb{X}$ and that we have derived a new point $x' \in \mathbb{X}$ from it using the unary search operator.
- ΔE then be the difference between the objective value of x' and x:

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- $\Delta E < 0$ means that the new x' is better than x since $f(\gamma(x')) < f(\gamma(x))$.
- $\Delta E > 0$ means that the new solution is worse.
- $\Delta E = 0$ means that both have the same quality.

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \left\{ \right. \tag{2}$$

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \begin{cases} 1 \\ \\ \end{cases} \tag{2}$$

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \begin{cases} 1 \quad \text{if } \Delta E \leq 0 \\ \end{cases} \tag{2}$$

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \end{cases} \tag{2}$$

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \end{cases} \tag{2}$$

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \\ 0 & \end{cases} \tag{2}$$

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \land T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T = 0) \end{cases} \tag{2}$$

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T = 0) \end{cases} \tag{2}$$

- If the new point x' is better than the current point x, i.e., $\Delta E < 0$, then we will definitely accept it.

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T = 0) \end{cases} \tag{2}$$

- If the new point x' is better (or, at least, not worse) than the current point x, i.e., $\Delta E \leq 0$, then we will definitely accept it.

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \land T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T = 0) \end{cases} \tag{2}$$

- If the new point x' is better (or, at least, not worse) than the current point x, i.e., $\Delta E \leq 0$, then we will definitely accept it.
- If the new point x' is worse ($\Delta E > 0$), then the acceptance probability

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T = 0) \end{cases} \tag{2}$$

- If the new point x' is better (or, at least, not worse) than the current point x, i.e., $\Delta E \leq 0$, then we will definitely accept it.
- If the new point x' is worse ($\Delta E > 0$), then the acceptance probability
  1. gets smaller the larger ΔE is.

Acceptance Probability

$$\Delta E = f(\gamma(x')) - f(\gamma(x)) \tag{1}$$

- The probability P to accept the new solution x' (and discard the current one x) is:

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T = 0) \end{cases} \tag{2}$$

- If the new point x' is better (or, at least, not worse) than the current point x, i.e., $\Delta E \leq 0$, then we will definitely accept it.
- If the new point x' is worse ($\Delta E > 0$), then the acceptance probability
  1. gets smaller the larger ΔE is and
  2. gets smaller the smaller the so-called "temperature" $T \geq 0$ is.

Ingredient: Temperature Schedule

Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T = 0) \end{cases} \tag{2}$$

Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T = 0) \end{cases} \tag{2}$$

- What about this temperature T?

Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T = 0) \end{cases} \tag{2}$$

- What about this temperature T?
- The temperature is defined to decrease and approaches zero with a rising number τ of algorithm iterations, i.e., the performed objective function evaluations.

Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T = 0) \end{cases} \quad (2)$$

- What about this temperature T?
- The temperature is defined to decrease and approaches zero with a rising number τ of performed objective function evaluations.
- The optimization process is initially "hot" and T is high.

Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T = 0) \end{cases} \tag{2}$$

- What about this temperature T?
- The temperature is defined to decrease and approaches zero with a rising number τ of performed objective function evaluations.
- The optimization process is initially "hot" and T is high.
- Then, even significantly worse solutions may be accepted.

Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \land T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T = 0) \end{cases} \tag{2}$$

- What about this temperature T?
- The temperature is defined to decrease and approaches zero with a rising number τ of performed objective function evaluations.
- The optimization process is initially "hot" and T is high.
- Then, even significantly worse solutions may be accepted.
- Over time, the process "cools" down and T decreases.

Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \land T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T = 0) \end{cases} \tag{2}$$

- What about this temperature T?
- The temperature is defined to decrease and approaches zero with a rising number τ of performed objective function evaluations.
- The optimization process is initially "hot" and T is high.
- Then, even significantly worse solutions may be accepted.
- Over time, the process "cools" down and T decreases.
- Slowly, fewer and fewer worse solutions are accepted and more likely such which are only a bit worse.

Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \wedge T > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T = 0) \end{cases} \quad (2)$$

- What about this temperature T?
- The temperature is defined to decrease and approaches zero with a rising number τ of performed objective function evaluations.
- The optimization process is initially "hot" and T is high.
- Then, even significantly worse solutions may be accepted.
- Over time, the process "cools" down and T decreases.
- Slowly, fewer and fewer worse solutions are accepted and more likely such which are only a bit worse.
- At temperature $T = 0$, the algorithm only accepts better solutions.

Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T(\tau)}} & \text{if } \Delta E > 0 \wedge T(\tau) > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T(\tau) = 0) \end{cases} \tag{2}$$

- What about this temperature $T(\tau)$?
- The temperature is defined to decrease and approaches zero with a rising number τ of performed objective function evaluations.
- The optimization process is initially "hot" and $T(\tau)$ is high.
- Then, even significantly worse solutions may be accepted.
- Over time, the process "cools" down and $T(\tau)$ decreases.
- Slowly, fewer and fewer worse solutions are accepted and more likely such which are only a bit worse.
- At temperature $T(\tau) = 0$, the algorithm only accepts better solutions.
- T is a monotonously decreasing function $T(\tau)$: the "temperature schedule."

Conditions for Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T(\tau)}} & \text{if } \Delta E > 0 \land T(\tau) > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0) \end{cases} \tag{2}$$

Conditions for Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T(\tau)}} & \text{if } \Delta E > 0 \land T(\tau) > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0) \end{cases} \tag{2}$$

- The temperature $T(\tau)$ is defined to decrease and approaches zero with a rising number τ of performed objective function evaluations.

Conditions for Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T(\tau)}} & \text{if } \Delta E > 0 \land T(\tau) > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0) \end{cases} \tag{2}$$

- The temperature $T(\tau)$ is defined to decrease and approaches zero with a rising number τ of performed objective function evaluations.
- It holds that $\lim_{\tau \to +\infty} T(\tau) = 0$.

Conditions for Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T(\tau)}} & \text{if } \Delta E > 0 \land T(\tau) > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0) \end{cases} \quad (2)$$

- The temperature $T(\tau)$ is defined to decrease and approaches zero with a rising number τ of performed objective function evaluations.
- It holds that $\lim_{\tau \to +\infty} T(\tau) = 0$.
- It begins with an start temperature T_s at $\tau = 1$.

Conditions for Temperature Schedule

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T(\tau)}} & \text{if } \Delta E > 0 \land T(\tau) > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0) \end{cases} \tag{2}$$

- The temperature $T(\tau)$ is defined to decrease and approaches zero with a rising number τ of performed objective function evaluations.
- It holds that $\lim_{\tau \to +\infty} T(\tau) = 0$.
- It begins with an start temperature T_s at $\tau = 1$.
- Apart from this, we can define $T(\tau)$ in any way we want.

Base Class for Implementing Temperature Schedules

```java
package aitoa.algorithms;

public abstract class TemperatureSchedule {
// unnecessary things omitted here
  public final double startTemperature; // ≡ T_s

  public abstract double temperature(long tau); // ≡ T(τ)

}
```

Exponential Temperature Schedule

- In an exponential temperature schedule, the temperature decreases exponentially with time (as the name implies).

Exponential Temperature Schedule

- In an exponential temperature schedule, the temperature decreases exponentially with time (as the name implies).
- Besides the start temperature T_s, it has a parameter $\varepsilon \in (0, 1)$ which tunes the speed of the temperature decrease.

Exponential Temperature Schedule

- In an exponential temperature schedule, the temperature decreases exponentially with time (as the name implies).
- Besides the start temperature T_s, it has a parameter $\varepsilon \in (0, 1)$ which tunes the speed of the temperature decrease.

$$T(\tau) = T_s * (1 - \varepsilon)^{\tau - 1} \tag{3}$$

Exponential Temperature Schedule

- In an exponential temperature schedule, the temperature decreases exponentially with time (as the name implies).
- Besides the start temperature T_s, it has a parameter $\varepsilon \in (0, 1)$ which tunes the speed of the temperature decrease.

$$T(\tau) = T_s * (1 - \varepsilon)^{\tau - 1} \tag{3}$$

- Higher values of ε lead to a faster temperature decline.

Exponential Temperature Schedule

```java
package aitoa.algorithms;

public class Exponential extends TemperatureSchedule {
// unnecessary things omitted here
  public final double epsilon; // ≡ ε

  public double temperature(long tau) {
    // T(τ) = T_s * (1 − ε)^(τ−1)
    return (this.startTemperature * Math.pow((1d -
        this.epsilon), (tau - 1L)));
  }
}
```

Logarithmic Temperature Schedule

- The logarithmic temperature schedule will prevent the temperature from becoming very small for a longer time.

Logarithmic Temperature Schedule

- The logarithmic temperature schedule will prevent the temperature from becoming very small for a longer time.
- Compared to the exponential schedule, it will longer retain a higher probability to accept worse solutions.

Logarithmic Temperature Schedule

- The logarithmic temperature schedule will prevent the temperature from becoming very small for a longer time.
- Compared to the exponential schedule, it will longer retain a higher probability to accept worse solutions.
- It also has the parameters $\varepsilon \in (0, \infty)$ and T_s.

Logarithmic Temperature Schedule

- The logarithmic temperature schedule will prevent the temperature from becoming very small for a longer time.
- Compared to the exponential schedule, it will longer retain a higher probability to accept worse solutions.
- It also has the parameters $\varepsilon \in (0, \infty)$ and T_s.

$$T(\tau) = \frac{T_s}{\ln\left(\varepsilon(\tau - 1) + e\right)} \tag{4}$$

Logarithmic Temperature Schedule

- The logarithmic temperature schedule will prevent the temperature from becoming very small for a longer time.
- Compared to the exponential schedule, it will longer retain a higher probability to accept worse solutions.
- It also has the parameters $\varepsilon \in (0, \infty)$ and T_s.

$$T(\tau) = \frac{T_s}{\ln\left(\varepsilon(\tau - 1) + e\right)} \tag{4}$$

- Larger values of ε again lead to a faster temperature decline.

Logarithmic Temperature Schedule

```java
package aitoa.algorithms;

public class Logarithmic extends TemperatureSchedule {
// unnecessary things omitted here
  public final double epsilon; // ≡ ε

  public double temperature(long tau) {
    // T(τ) = T_s / ln(ε(τ−1)+e)
    return (this.startTemperature / Math.log(((tau - 1L)
        * this.epsilon) + Math.E));
  }
}
```

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?
- Let's think back again about Evolutionary Algorithms[2, 8–11].

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?
- Let's think back again about Evolutionary Algorithms[2,8–11].
- By using the population size parameters μ and λ, we can tune the behavior of an EA between random sampling ($\mu \to \infty$ or $\lambda \to \infty$) and hill climbing ($\mu = \lambda = 1$).

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?
- Let's think back again about Evolutionary Algorithms[2, 8–11].
- By using the population size parameters μ and λ, we can tune the behavior of an EA between random sampling ($\mu \to \infty$ or $\lambda \to \infty$) and hill climbing ($\mu = \lambda = 1$).
- This allowed us to tune between exploration and exploitation, to find a "sweet spot" where the algorithm performs best.

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?
- Let's think back again about Evolutionary Algorithms[2, 8–11].
- By using the population size parameters μ and λ, we can tune the behavior of an EA between random sampling ($\mu \to \infty$ or $\lambda \to \infty$) and hill climbing ($\mu = \lambda = 1$).
- This allowed us to tune between exploration and exploitation, to find a "sweet spot" where the algorithm performs best.
- The temperature schedule in SA allows us to do the same.

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?
- Let's think back again about Evolutionary Algorithms[2, 8–11].
- By using the population size parameters μ and λ, we can tune the behavior of an EA between random sampling ($\mu \to \infty$ or $\lambda \to \infty$) and hill climbing ($\mu = \lambda = 1$).
- This allowed us to tune between exploration and exploitation, to find a "sweet spot" where the algorithm performs best.
- The temperature schedule in SA allows us to do the same but dynamically!

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?
- Let's think back again about Evolutionary Algorithms[2, 8–11].
- By using the population size parameters μ and λ, we can tune the behavior of an EA between random sampling ($\mu \to \infty$ or $\lambda \to \infty$) and hill climbing ($\mu = \lambda = 1$).
- This allowed us to tune between exploration and exploitation, to find a "sweet spot" where the algorithm performs best.
- The temperature schedule in SA allows us to do the same but dynamically!
- If T is high at the beginning. . .

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?
- Let's think back again about Evolutionary Algorithms[2, 8–11].
- By using the population size parameters μ and λ, we can tune the behavior of an EA between random sampling ($\mu \to \infty$ or $\lambda \to \infty$) and hill climbing ($\mu = \lambda = 1$).
- This allowed us to tune between exploration and exploitation, to find a "sweet spot" where the algorithm performs best.
- The temperature schedule in SA allows us to do the same but dynamically!
- If T is high at the beginning $\Rightarrow$ many bad solutions are accepted.

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?
- Let's think back again about Evolutionary Algorithms[2, 8–11].
- By using the population size parameters μ and λ, we can tune the behavior of an EA between random sampling ($\mu \to \infty$ or $\lambda \to \infty$) and hill climbing ($\mu = \lambda = 1$).
- This allowed us to tune between exploration and exploitation, to find a "sweet spot" where the algorithm performs best.
- The temperature schedule in SA allows us to do the same but dynamically!
- If T is high at the beginning $\Rightarrow$ many bad solutions are accepted $\Rightarrow$ random sampling.

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?
- Let's think back again about Evolutionary Algorithms[2, 8–11].
- By using the population size parameters μ and λ, we can tune the behavior of an EA between random sampling ($\mu \to \infty$ or $\lambda \to \infty$) and hill climbing ($\mu = \lambda = 1$).
- This allowed us to tune between exploration and exploitation, to find a "sweet spot" where the algorithm performs best.
- The temperature schedule in SA allows us to do the same but dynamically!
- If T is high at the beginning $\Rightarrow$ many bad solutions are accepted $\Rightarrow$ random sampling.
- At the end, $T \approx 0 \ldots$

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?
- Let's think back again about Evolutionary Algorithms[2,8–11].
- By using the population size parameters μ and λ, we can tune the behavior of an EA between random sampling ($\mu \to \infty$ or $\lambda \to \infty$) and hill climbing ($\mu = \lambda = 1$).
- This allowed us to tune between exploration and exploitation, to find a "sweet spot" where the algorithm performs best.
- The temperature schedule in SA allows us to do the same but dynamically!
- If T is high at the beginning $\Rightarrow$ many bad solutions are accepted $\Rightarrow$ random sampling.
- At the end, $T \approx 0 \Rightarrow$ no worse solutions are accepted anymore.

The Meaning of the Temperature Schedule

- Why do we have such a strange thing like a temperature schedule?
- Let's think back again about Evolutionary Algorithms[2, 8–11].
- By using the population size parameters μ and λ, we can tune the behavior of an EA between random sampling ($\mu \to \infty$ or $\lambda \to \infty$) and hill climbing ($\mu = \lambda = 1$).
- This allowed us to tune between exploration and exploitation, to find a "sweet spot" where the algorithm performs best.
- The temperature schedule in SA allows us to do the same but dynamically!
- If T is high at the beginning $\Rightarrow$ many bad solutions are accepted $\Rightarrow$ random sampling.
- At the end, $T \approx 0 \Rightarrow$ no worse solutions are accepted anymore $\Rightarrow$ hill climbing.

Algorithm Implementation

Simulated Annealing Algorithm

- Simulated Annealing = Hill Climbing + probabilistically accepting worse solutions

Simulated Annealing Algorithm

- Simulated Annealing = Hill Climbing + probabilistically accepting worse solutions
- Simple Concept

Simulated Annealing Algorithm

- Simulated Annealing = Hill Climbing + probabilistically accepting worse solutions
- Simple Concept:
    1. Start with $\tau = 1$.

Simulated Annealing Algorithm

- Simulated Annealing = Hill Climbing + probabilistically accepting worse solutions
- Simple Concept:
  1. Start with $\tau = 1$.
  2. Create random initial point x, which also becomes the first "current point" x and the overall best point x_b.

Simulated Annealing Algorithm

- Simulated Annealing = Hill Climbing + probabilistically accepting worse solutions
- Simple Concept:
  1. Start with $\tau = 1$.
  2. Create random initial point x, which also becomes the first "current point" x and the overall best point x_b.
  3. Create a modified copy x' of the current point x.

Simulated Annealing Algorithm

- Simulated Annealing = Hill Climbing + probabilistically accepting worse solutions
- Simple Concept:
  1. Start with $\tau = 1$.
  2. Create random initial point x, which also becomes the first "current point" x and the overall best point x_b.
  3. Create a modified copy x' of the current point x.
  4. Set $\tau = \tau + 1$.

Simulated Annealing Algorithm

- Simulated Annealing = Hill Climbing + probabilistically accepting worse solutions
- Simple Concept:
  1. Start with $\tau = 1$.
  2. Create random initial point x, which also becomes the first "current point" x and the overall best point x_b.
  3. Create a modified copy x' of the current point x.
  4. Set $\tau = \tau + 1$.
  5. If the new point x' is better than x_b, set $x_b = x'$.

Simulated Annealing Algorithm

- Simulated Annealing = Hill Climbing + probabilistically accepting worse solutions
- Simple Concept:
  1. Start with $\tau = 1$.
  2. Create random initial point x, which also becomes the first "current point" x and the overall best point x_b.
  3. Create a modified copy x' of the current point x.
  4. Set $\tau = \tau + 1$.
  5. If the new point x' is better than x_b, set $x_b = x'$.
  6. If the new point x' is better than x, set $x = x'$.

Simulated Annealing Algorithm

- Simulated Annealing = Hill Climbing + probabilistically accepting worse solutions
- Simple Concept:
  1. Start with $\tau = 1$.
  2. Create random initial point x, which also becomes the first "current point" x and the overall best point x_b.
  3. Create a modified copy x' of the current point x.
  4. Set $\tau = \tau + 1$.
  5. If the new point x' is better than x_b, set $x_b = x'$.
  6. If the new point x' is better than x, set $x = x'$.
  7. If it is worse ($\Delta E > 0$): accept it as current solution with probability $P(\Delta E, \tau)$ (which gets smaller over time and also the smaller the worse the new solution is) or otherwise reject it.

Simulated Annealing Algorithm

- Simulated Annealing = Hill Climbing + probabilistically accepting worse solutions
- Simple Concept:
  1. Start with $\tau = 1$.
  2. Create random initial point x, which also becomes the first "current point" x and the overall best point x_b.
  3. Create a modified copy x' of the current point x.
  4. Set $\tau = \tau + 1$.
  5. If the new point x' is better than x_b, set $x_b = x'$.
  6. If the new point x' is better than x, set $x = x'$.
  7. If it is worse ($\Delta E > 0$): accept it as current solution with probability $P(\Delta E, \tau)$ (which gets smaller over time and also the smaller the worse the new solution is) or otherwise reject it.
  8. Go back to 3. (until the time is up)

Simulated Annealing Algorithm

- Simulated Annealing = Hill Climbing + probabilistically accepting worse solutions
- Simple Concept:
  1. Start with $\tau = 1$.
  2. Create random initial point x, which also becomes the first "current point" x and the overall best point x_b.
  3. Create a modified copy x' of the current point x.
  4. Set $\tau = \tau + 1$.
  5. If the new point x' is better than x_b, set $x_b = x'$.
  6. If the new point x' is better than x, set $x = x'$.
  7. If it is worse $(\Delta E > 0)$: accept it as current solution with probability $P(\Delta E, \tau)$ (which gets smaller over time and also the smaller the worse the new solution is) or otherwise reject it.
  8. Go back to 3. (until the time is up)
  4. Return the best ever-encountered point x_b.

Implementing Simulated Annealing

```java
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> {
// unnecessary things omitted
//
//
//
//

//
//
//
//

//
//
//
//
//
//
//
//
//
//
//
//
}
```

Implementing Simulated Annealing

```
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
//
//
//

//
//
//
//

//
//
//
//
//
//
//
//
//
//
//
  }
}
```

Implementing Simulated Annealing

```java
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
//
//

//
//
//
//

//
//
//
//
//
//
//
//
//
//
//
  }
}
```

Implementing Simulated Annealing

```java
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
//

//
//
//
//

//
//
//
//
//
//
//
//
//
//
//
  }
}
```

Implementing Simulated Annealing

```java
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

//
//
//
//

//
//
//
//
//
//
//
//
//
//
//
  }
}
```

Implementing Simulated Annealing

```java
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

// create starting point: a random point in the search space
    this.nullary.apply(xCur, random);     // put random point in xCur
//
//

//
//
//
//
//
//
//
//
//
//
//
  }
}
```

Implementing Simulated Annealing

```
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

// create starting point: a random point in the search space
    this.nullary.apply(xCur, random);     // put random point in xCur
    double fCur = process.evaluate(xCur); // map xCur to 𝕐 and evaluate objective f
//

//
//
//
//
//
//
//
//
//
//
//
  } // process will have automatically remembered the best candidate solution
}
```

Implementing Simulated Annealing

```
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

// create starting point: a random point in the search space
    this.nullary.apply(xCur, random);     // put random point in xCur
    double fCur = process.evaluate(xCur); // map xCur to 𝕐 and evaluate objective f
    long   tau  = 1L;                     // initialize step counter to 1

//
//
//
//
//
//
//
//
//
//
//
  } // process will have automatically remembered the best candidate solution
}
```

Implementing Simulated Annealing

```java
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

// create starting point: a random point in the search space
    this.nullary.apply(xCur, random);     // put random point in xCur
    double fCur = process.evaluate(xCur); // map xCur to 𝕐 and evaluate objective f
    long   tau  = 1L;                     // initialize step counter to 1

//
    this.unary.apply(xCur, xNew, random); // create modified copy xNew of xCur
//
//
//
//
//
//
//
//
//
  } // process will have automatically remembered the best candidate solution
}
```

Implementing Simulated Annealing

```java
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

// create starting point: a random point in the search space
    this.nullary.apply(xCur, random);     // put random point in xCur
    double fCur = process.evaluate(xCur); // map xCur to 𝕐 and evaluate objective f
    long   tau  = 1L;                     // initialize step counter to 1

//
    this.unary.apply(xCur, xNew, random); // create modified copy xNew of xCur
    ++tau; // increase step counter
//
//
//
//
//
//
//
//
  } // process will have automatically remembered the best candidate solution
}
```

Implementing Simulated Annealing

```java
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

// create starting point: a random point in the search space
    this.nullary.apply(xCur, random);     // put random point in xCur
    double fCur = process.evaluate(xCur); // map xCur to 𝕐 and evaluate objective f
    long   tau  = 1L;                     // initialize step counter to 1

//
    this.unary.apply(xCur, xNew, random); // create modified copy xNew of xCur
    ++tau; // increase step counter
    double fNew = process.evaluate(xNew); // map xNew from 𝕏 to 𝕐 and evaluate result
//
//
//
//
//
//
//
  } // process will have automatically remembered the best candidate solution
}
```

Implementing Simulated Annealing

```java
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

// create starting point: a random point in the search space
    this.nullary.apply(xCur, random);     // put random point in xCur
    double fCur = process.evaluate(xCur); // map xCur to 𝕐 and evaluate objective f
    long   tau  = 1L;                     // initialize step counter to 1

//
    this.unary.apply(xCur, xNew, random); // create modified copy xNew of xCur
    ++tau; // increase step counter
    double fNew = process.evaluate(xNew); // map xNew from 𝕏 to 𝕐 and evaluate result
    if  (fNew <= fCur) {  // accept if new solution is better solution
//
//
//
//
    } // otherwise fNew > fCur and not accepted
//
  } // process will have automatically remembered the best candidate solution
}
```

Implementing Simulated Annealing

```
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

// create starting point: a random point in the search space
    this.nullary.apply(xCur, random);     // put random point in xCur
    double fCur = process.evaluate(xCur); // map xCur to 𝕐 and evaluate objective f
    long   tau  = 1L;                     // initialize step counter to 1

//
    this.unary.apply(xCur, xNew, random); // create modified copy xNew of xCur
    ++tau; // increase step counter
    double fNew = process.evaluate(xNew); // map xNew from 𝕏 to 𝕐 and evaluate result
    if ((fNew <= fCur) || // accept if new solution is better solution OR
        (random.nextDouble() < // probability is exp(−ΔE/T) using −ΔE = −(fNew − fCur)
          Math.exp((fCur - fNew) / this.schedule.temperature(tau)))) {
//
//
    } // otherwise fNew > fCur and not accepted
//
  } // process will have automatically remembered the best candidate solution
}
```

Implementing Simulated Annealing

```
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

// create starting point: a random point in the search space
    this.nullary.apply(xCur, random);     // put random point in xCur
    double fCur = process.evaluate(xCur); // map xCur to 𝕐 and evaluate objective f
    long   tau  = 1L;                     // initialize step counter to 1

//
    this.unary.apply(xCur, xNew, random); // create modified copy xNew of xCur
    ++tau; // increase step counter
    double fNew = process.evaluate(xNew); // map xNew from 𝕏 to 𝕐 and evaluate result
    if ((fNew <= fCur) || // accept if new solution is better solution OR
        (random.nextDouble() < // probability is exp(−ΔE/T) using −ΔE = −(fNew − fCur)
           Math.exp((fCur - fNew) / this.schedule.temperature(tau)))) {
      fCur = fNew; // update current objective value
//
    } // otherwise fNew > fCur and not accepted
//
  } // process will have automatically remembered the best candidate solution
}
```

Implementing Simulated Annealing

```java
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

// create starting point: a random point in the search space
    this.nullary.apply(xCur, random);     // put random point in xCur
    double fCur = process.evaluate(xCur); // map xCur to 𝕐 and evaluate objective f
    long   tau  = 1L;                     // initialize step counter to 1

//
    this.unary.apply(xCur, xNew, random); // create modified copy xNew of xCur
    ++tau; // increase step counter
    double fNew = process.evaluate(xNew); // map xNew from 𝕏 to 𝕐 and evaluate result
    if ((fNew <= fCur) || // accept if new solution is better solution OR
        (random.nextDouble() < // probability is exp(−ΔE/T) using −ΔE = −(fNew − fCur)
          Math.exp((fCur - fNew) / this.schedule.temperature(tau)))) {
      fCur = fNew; // update current objective value
      process.getSearchSpace().copy(xNew, xCur); // copy xNew to xCur
    } // otherwise fNew > fCur and not accepted
//
  } // process will have automatically remembered the best candidate solution
}
```

Implementing Simulated Annealing

```java
package aitoa.algorithms;

public class SimulatedAnnealing<X, Y> extends Metaheuristic1<X, Y> {
// unnecessary things omitted
  public void solve(IBlackBoxProcess<X, Y> process) {
    X xNew = process.getSearchSpace().create();
    X xCur = process.getSearchSpace().create();
    Random random = process.getRandom();// get random number generator

// create starting point: a random point in the search space
    this.nullary.apply(xCur, random);     // put random point in xCur
    double fCur = process.evaluate(xCur); // map xCur to 𝕐 and evaluate objective f
    long   tau  = 1L;                     // initialize step counter to 1

    do { // repeat until budget exhausted
      this.unary.apply(xCur, xNew, random); // create modified copy xNew of xCur
      ++tau; // increase step counter
      double fNew = process.evaluate(xNew); // map xNew from 𝕏 to 𝕐 and evaluate result
      if ((fNew <= fCur) || // accept if new solution is better solution OR
          (random.nextDouble() < // probability is exp(−ΔE/T) using −ΔE = −(fNew − fCur)
             Math.exp((fCur - fNew) / this.schedule.temperature(tau)))) {
        fCur = fNew; // update current objective value
        process.getSearchSpace().copy(xNew, xCur); // copy xNew to xCur
      } // otherwise fNew > fCur and not accepted
    } while (!process.shouldTerminate()); // until time is up
  } // process will have automatically remembered the best candidate solution
}
```

Configuring the Algorithm

Configuring the Algorithm

- Our algorithm has four parameters.

Configuring the Algorithm

- Our algorithm has four parameters:
    1. the start temperature T_s

Configuring the Algorithm

- Our algorithm has four parameters:
    1. the start temperature T_s,
    2. the parameter ε

Configuring the Algorithm

- Our algorithm has four parameters:
    1. the start temperature T_s,
    2. the parameter ε,
    3. the type of temperature schedule to use (here, logarithmic or exponential)

Configuring the Algorithm

- Our algorithm has four parameters:
  1. the start temperature T_s,
  2. the parameter ε,
  3. the type of temperature schedule to use (here, logarithmic or exponential), and
  4. the unary search operator (in our case, we could use `1swap` or `nswap`).

Configuring the Algorithm

- Our algorithm has four parameters:
  1. the start temperature T_s,
  2. the parameter ε,
  3. the type of temperature schedule to use (here, logarithmic or exponential), and
  4. the unary search operator (in our case, we could use `1swap` or `nswap`).
- We will only use `1swap` as choice for the unary operator and focus on the exponential temperature schedule.

Configuring the Algorithm

- Our algorithm has four parameters:
  1. the start temperature T_s,
  2. the parameter ε,
  3. the type of temperature schedule to use (here, logarithmic or exponential), and
  4. the unary search operator (in our case, we could use `1swap` or `nswap`).
- We will only use `1swap` as choice for the unary operator and focus on the exponential temperature schedule.
- This leaves T_s and ε to be configured.

Configuring the Algorithm

- Our algorithm has four parameters:
  1. the start temperature T_s,
  2. the parameter ε,
  3. the type of temperature schedule to use (here, logarithmic or exponential), and
  4. the unary search operator (in our case, we could use `1swap` or `nswap`).
- We will only use `1swap` as choice for the unary operator and focus on the exponential temperature schedule.
- This leaves T_s and ε to be configured.
- Interestingly, we may be able to very roughly compute some reasonable values for them!

Simulated Annealing as Improved Hill Climber

- Let us consider Simulated Annealing as an improved Hill Climber.

Simulated Annealing as Improved Hill Climber

- Let us consider Simulated Annealing as an improved Hill Climber and look at the experimental results of this algorithm.

Simulated Annealing as Improved Hill Climber

- Let us consider Simulated Annealing as an improved Hill Climber and look at the experimental results of this algorithm.

$\mathcal{I}$	med(total FEs)	sd
`abz7`	35'648'639	28
`la24`	70'952'285	56
`swv15`	21'662'286	137
`yn4`	27'090'511	48
median	**31'369'575**	**52**

Simulated Annealing as Improved Hill Climber

- Let us consider Simulated Annealing as an improved Hill Climber and look at the experimental results of this algorithm.

$\mathcal{I}$	med(total FEs)	sd
abz7	35'648'639	28
la24	70'952'285	56
swv15	21'662'286	137
yn4	27'090'511	48
median	**31'369'575**	**52**

- hc_1swap performs 30 million FEs (within the three minute budget) in median over all instances.

Simulated Annealing as Improved Hill Climber

- Let us consider Simulated Annealing as an improved Hill Climber and look at the experimental results of this algorithm.

$\mathcal{I}$	med(total FEs)	sd
abz7	35'648'639	28
la24	70'952'285	56
swv15	21'662'286	137
yn4	27'090'511	48
median	**31'369'575**	**52**

- hc_1swap performs 30 million FEs (within the three minute budget) in median over all instances.
- The median of the standard deviations of the result quality at the end of the three minutes (over all instances) is about 50.

Simulated Annealing as Improved Hill Climber

- Let us consider Simulated Annealing as an improved Hill Climber and look at the experimental results of this algorithm.

$\mathcal{I}$	med(total FEs)	sd
`abz7`	35'648'639	28
`la24`	70'952'285	56
`swv15`	21'662'286	137
`yn4`	27'090'511	48
median	**31'369'575**	**52**

- `hc_1swap` performs 30 million FEs (within the three minute budget) in median over all instances.
- The median of the standard deviations of the result quality at the end of the three minutes (over all instances) is about 50.
- What can we do with these information?

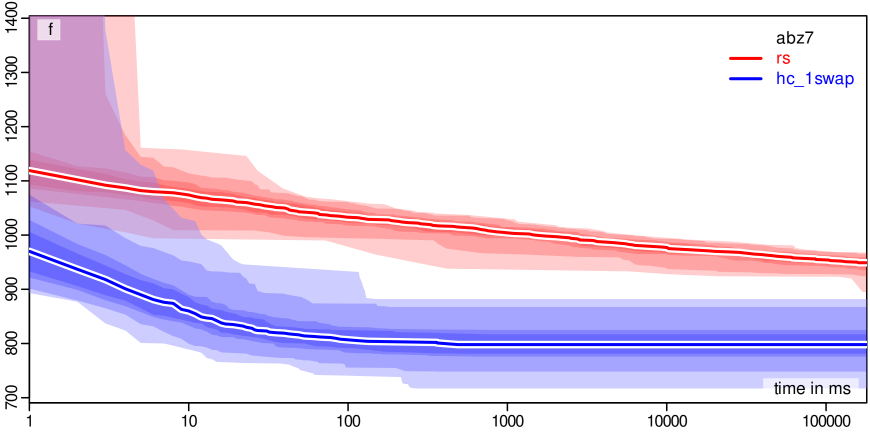
End Result Standard Deviation

- The median standard deviation of the final results of `hc_1swap` of 50 tells us something about the local optima.

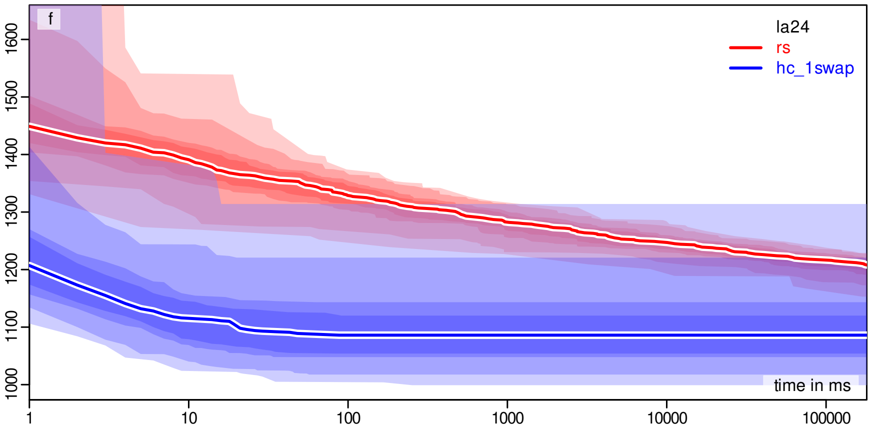
End Result Standard Deviation

- The median standard deviation of the final results of `hc_1swap` of 50 tells us something about the local optima.
- We know that `hc_1swap` gets stuck in local optima – it stopped improving after just one second!

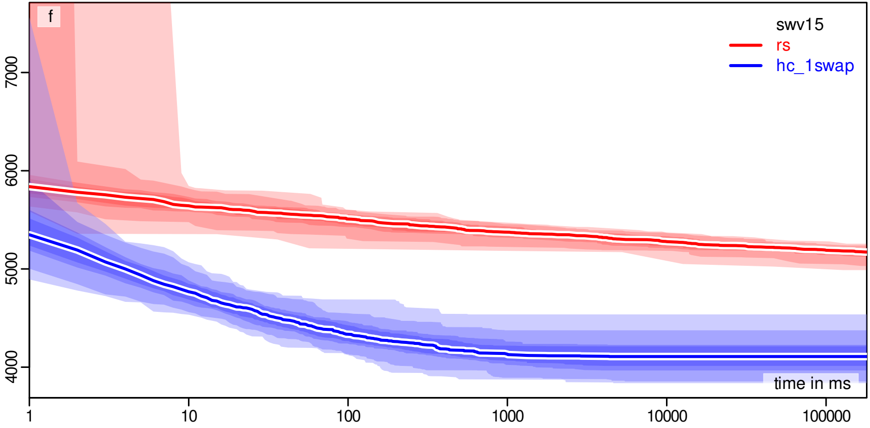
End Result Standard Deviation



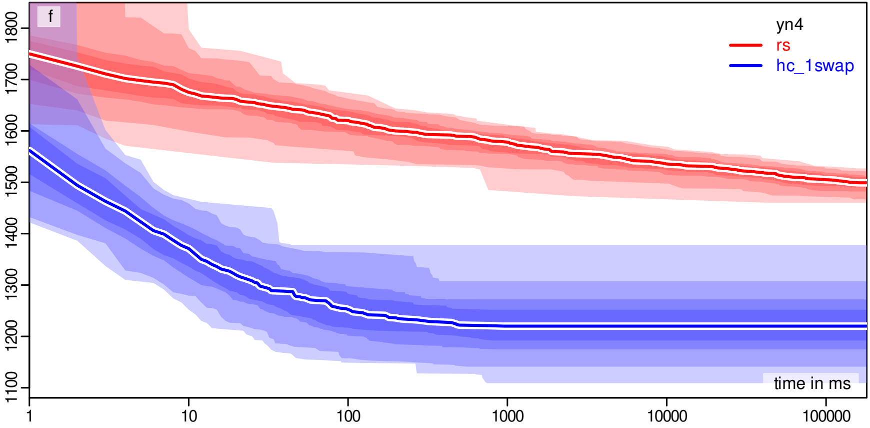
End Result Standard Deviation



End Result Standard Deviation



End Result Standard Deviation



End Result Standard Deviation

- The median standard deviation of the final results of `hc_1swap` of 50 tells us something about the local optima.
- We know that `hc_1swap` gets stuck in local optima – it stopped improving after just one second!
- The standard measures how spread out the local optima.

End Result Standard Deviation

- The median standard deviation of the final results of `hc_1swap` of 50 tells us something about the local optima.
- We know that `hc_1swap` gets stuck in local optima – it stopped improving after just one second!
- The standard measures how spread out the local optima.
- It is a gives us a good impression of how different the qualities of the local optima are that we can expect to see.

End Result Standard Deviation

- The median standard deviation of the final results of `hc_1swap` of 50 tells us something about the local optima.
- We know that `hc_1swap` gets stuck in local optima – it stopped improving after just one second!
- The standard measures how spread out the local optima.
- It is a gives us a good impression of how different the qualities of the local optima are that we can expect to see.
- Thus, accepting a solution which is worse by 50 units of makespan, i.e., with $\Delta E \approx 50$, should be possible at the beginning of the optimization process.

End Result Standard Deviation

- The median standard deviation of the final results of `hc_1swap` of 50 tells us something about the local optima.
- We know that `hc_1swap` gets stuck in local optima – it stopped improving after just one second!
- The standard measures how spread out the local optima.
- It is a gives us a good impression of how different the qualities of the local optima are that we can expect to see.
- Thus, accepting a solution which is worse by 50 units of makespan, i.e., with $\Delta E \approx 50$, should be possible at the beginning of the optimization process.
- Let's say that the probability to accept such a solution should be 10

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T(\tau)}} & \text{if } \Delta E > 0 \land T(\tau) > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0) \end{cases} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T(\tau)}} & \text{if } \Delta E > 0 \land T(\tau) > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0) \end{cases} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$P = e^{-\frac{\Delta E}{T(\tau)}} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$P = e^{-\frac{\Delta E}{T(\tau)}} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$P_{50} = e^{-\frac{\Delta E}{T(\tau)}} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$0.1 = e^{-\frac{\Delta E}{T(\tau)}} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$0.1 = e^{-\frac{\Delta E}{T(\tau)}} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$0.1 = e^{-\frac{50}{T(\tau)}} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$0.1 = e^{-\frac{50}{T(\tau)}} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$0.1 = e^{-\frac{50}{T_s}} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$0.1 = e^{-\frac{50}{T_s}} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$\ln 0.1 = -\frac{50}{T_s} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$\ln 0.1 = -\frac{50}{T_s} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$T_s = -\frac{50}{\ln 0.1} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$T_s = -\frac{50}{\ln 0.1} \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$T_s \approx 21.714\,724\,095 \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$T_s \approx 20 \tag{5}$$

From End Result Standard Deviation to Start Temperature

- The median standard deviation of the end result quality of the hill climber is 50.
- We want to accept a solution with $\Delta E = 50$ with probability $P_{50} = 0.1$ at $\tau = 1$.
- At $\tau = 1$, the temperature of any temperature schedule equals the start temperature T_s.
- We can solve the probability Equation 2 for T_s:

$$T_s \approx 20 \tag{5}$$

- A start temperature T_s of about 20 seems to be a good choice.

End Temperature

- Let us first think about the end temperature T_e that should be reached at the end of the run.

End Temperature

- Let us first think about the end temperature T_e that should be reached at the end of the run.
- While we know that $\lim_{\tau \to +\infty} T(\tau) = 0$, we also know that three minutes of runtime is less than $+\infty$.

End Temperature

- Let us first think about the end temperature T_e that should be reached at the end of the run.
- While we know that $\lim_{\tau \to +\infty} T(\tau) = 0$, we also know that three minutes of runtime is less than $+\infty$.
- $T = 0$ will thus not be reached within a finite number τ of steps and the actual end temperature T_e should probably be slightly above 0.

End Temperature

- Let us first think about the end temperature T_e that should be reached at the end of the run.
- While we know that $\lim_{\tau \to +\infty} T(\tau) = 0$, we also know that three minutes of runtime is less than $+\infty$.
- $T = 0$ will thus not be reached within a finite number τ of steps and the actual end temperature T_e should probably be slightly above 0.
- Let us remember back our `hcr_L_1swap` algorithm, i.e., the `1swap` hill climber restarting after L unsuccessful steps.

End Temperature

- Let us first think about the end temperature T_e that should be reached at the end of the run.
- While we know that $\lim_{\tau \to +\infty} T(\tau) = 0$, we also know that three minutes of runtime is less than $+\infty$.
- $T = 0$ will thus not be reached within a finite number τ of steps and the actual end temperature T_e should probably be slightly above 0.
- Let us remember back our `hcr_L_1swap` algorithm, i.e., the `1swap` hill climber restarting after L unsuccessful steps.
- There, we found $L = 2^{14} = 16'384$ to be reasonable choice.

End Temperature

- Let us first think about the end temperature T_e that should be reached at the end of the run.
- While we know that $\lim_{\tau \to +\infty} T(\tau) = 0$, we also know that three minutes of runtime is less than $+\infty$.
- $T = 0$ will thus not be reached within a finite number τ of steps and the actual end temperature T_e should probably be slightly above 0.
- Let us remember back our `hcr_L_1swap` algorithm, i.e., the `1swap` hill climber restarting after L unsuccessful steps.
- There, we found $L = 2^{14} = 16'384$ to be reasonable choice.
- As idea to get a reasonable T_e, we could say that the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

End Temperature

- Let us first think about the end temperature T_e that should be reached at the end of the run.
- While we know that $\lim_{\tau\to+\infty} T(\tau) = 0$, we also know that three minutes of runtime is less than $+\infty$.
- $T = 0$ will thus not be reached within a finite number τ of steps and the actual end temperature T_e should probably be slightly above 0.
- Let us remember back our `hcr_L_1swap` algorithm, i.e., the `1swap` hill climber restarting after L unsuccessful steps.
- There, we found $L = 2^{14} = 16'384$ to be reasonable choice.
- As idea to get a reasonable T_e, we could say that the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.
- Then, the chance to accept a solution marginally worse than the current one would be about as large as making a complete restart in `hcr_16384_1swap`.

End Temperature

- Let us first think about the end temperature T_e that should be reached at the end of the run.
- While we know that $\lim_{\tau \to +\infty} T(\tau) = 0$, we also know that three minutes of runtime is less than $+\infty$.
- $T = 0$ will thus not be reached within a finite number τ of steps and the actual end temperature T_e should probably be slightly above 0.
- Let us remember back our `hcr_L_1swap` algorithm, i.e., the `1swap` hill climber restarting after L unsuccessful steps.
- There, we found $L = 2^{14} = 16'384$ to be reasonable choice.
- As idea to get a reasonable T_e, we could say that the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.
- Then, the chance to accept a solution marginally worse than the current one would be about as large as making a complete restart in `hcr_16384_1swap`.
- This is a bit far fetched.

End Temperature

- Let us first think about the end temperature T_e that should be reached at the end of the run.
- While we know that $\lim_{\tau \to +\infty} T(\tau) = 0$, we also know that three minutes of runtime is less than $+\infty$.
- $T = 0$ will thus not be reached within a finite number τ of steps and the actual end temperature T_e should probably be slightly above 0.
- Let us remember back our `hcr_L_1swap` algorithm, i.e., the `1swap` hill climber restarting after L unsuccessful steps.
- There, we found $L = 2^{14} = 16'384$ to be reasonable choice.
- As idea to get a reasonable T_e, we could say that the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.
- Then, the chance to accept a solution marginally worse than the current one would be about as large as making a complete restart in `hcr_16384_1swap`.
- This is a bit far fetched, but as a rough guess it will do.

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T(\tau)}} & \text{if } \Delta E > 0 \wedge T(\tau) > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \wedge T(\tau) = 0) \end{cases} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$P = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T(\tau)}} & \text{if } \Delta E > 0 \land T(\tau) > 0 \\ 0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0) \end{cases} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$P = e^{-\frac{\Delta E}{T(\tau)}} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$P = e^{-\frac{\Delta E}{T(\tau)}} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$P_e = e^{-\frac{\Delta E}{T(\tau)}} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$\frac{1}{16'384} = e^{-\frac{\Delta E}{T(\tau)}} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$\frac{1}{16'384} = e^{-\frac{\Delta E}{T(\tau)}} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$\frac{1}{16'384} = e^{-\frac{\Delta E}{T_e}} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$\frac{1}{16'384} = e^{-\frac{\Delta E}{T_e}} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$\frac{1}{16'384} = e^{-\frac{1}{T_e}} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$\frac{1}{16'384} = e^{-\frac{1}{T_e}} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$\ln \frac{1}{16'384} = -\frac{1}{T_e} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$\ln \frac{1}{16'384} = -\frac{1}{T_e} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$T_e = -\frac{1}{\ln \frac{1}{16'384}} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$T_e = -\frac{1}{\ln \frac{1}{16'384}} \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$T_e \approx 0.103\,049\,646 \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$T_e \approx 0.1 \tag{6}$$

End Temperature

- To get an end temperature T_e, the end probability P_e to accept a solution which is $\Delta E = 1$ makespan unit worse than the current solution should be $P_e = 1/L = \frac{1}{16'384}$ at the end of our Simulated Annealing runs.

$$T_e \approx 0.1 \tag{6}$$

- It seems that an end temperature $T_e \approx= 0.1$ is a reasonable setting for SA using `1swap`.

Epsilon from End Temperature and Iteration

- We now want to find a good setting for the ε parameter.

Epsilon from End Temperature and Iteration

- We now want to find a good setting for the ε parameter.
- This parameter plays a role in the exponential temperature schedule.

Epsilon from End Temperature and Iteration

- We now want to find a good setting for the ε parameter.
- This parameter plays a role in the exponential temperature schedule.

$$T(\tau) = T_s * (1 - \varepsilon)^{\tau - 1} \tag{3}$$

Epsilon from End Temperature and Iteration

- We now want to find a good setting for the ε parameter.
- This parameter plays a role in the exponential temperature schedule.
- It relates the temperature $T(\tau)$ at a given iteration τ to the iteration index τ.

$$T(\tau) = T_s * (1 - \varepsilon)^{\tau - 1} \tag{3}$$

Epsilon from End Temperature and Iteration

- We now want to find a good setting for the ε parameter.
- This parameter plays a role in the exponential temperature schedule.
- It relates the temperature $T(\tau)$ at a given iteration τ to the iteration index τ.
- In order to compute a rough guess for ε, we thus need a value for τ and one for $T(\tau)$ first.

$$T(\tau) = T_s * (1 - \varepsilon)^{\tau - 1} \tag{3}$$

Epsilon from End Temperature and Iteration

- We now want to find a good setting for the ε parameter.
- This parameter plays a role in the exponential temperature schedule.
- It relates the temperature $T(\tau)$ at a given iteration τ to the iteration index τ.
- In order to compute a rough guess for ε, we thus need a value for τ and one for $T(\tau)$ first.
- The start temperature T_s alone does not help us here, but we now also have an end temperature T_e.

$$T(\tau) = T_s * (1 - \varepsilon)^{\tau - 1} \tag{3}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$T(\tau) = T_s * (1 - \varepsilon)^{\tau - 1} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$T(\tau) = T_s * (1 - \varepsilon)^{\tau-1} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$T_e = T_s * (1 - \varepsilon)^{\tau - 1} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = T_s * (1 - \varepsilon)^{\tau - 1} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = T_s * (1 - \varepsilon)^{\tau - 1} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = T_s * (1 - \varepsilon)^{30'000'000 - 1} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = T_s * (1 - \varepsilon)^{30'000'000 - 1} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = T_s * (1 - \varepsilon)^{29'999'999} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = T_s * (1 - \varepsilon)^{29'999'999} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = 20 * (1 - \varepsilon)^{29'999'999} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = 20 * (1 - \varepsilon)^{29'999'999} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$\frac{0.1}{20} = (1 - \varepsilon)^{29'999'999} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.005 = (1 - \varepsilon)^{29'999'999} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.005 = (1 - \varepsilon)^{29'999'999} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.005^{1/29'999'999} = 1 - \varepsilon \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.999\,999\,823\,389 \approx 1 - \varepsilon \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$\varepsilon \approx 1 - 0.999\,999\,823\,389 \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$\varepsilon \approx 1 - 0.999\,999\,823\,389 \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$\varepsilon \approx 0.000\,000\,176\,610\,569 \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$\varepsilon \approx 1.776 * 10^{-7} \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$\varepsilon \in \left[1 * 10^{-7}, 2 * 10^{-7}\right] \tag{7}$$

Epsilon from End Temperature and Iteration

- We have a start temperature T_s and an end temperature T_e.
- What we need it we want to solve Equation 3 for ε is the iteration index τ at which $T(\tau) = T_e$.
- Before, we said that our optimization processes run for about 30'000'000 FEs in median.
- Since T_e is the end temperature, the right value for τ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$\varepsilon \in \left[1 * 10^{-7}, 2 * 10^{-7}\right] \tag{7}$$

- Values of ε between 1 and 2 times 10^{-7} seem reasonable.

Configuration from Previous Knowledge

- We now have reasonable parameter values for our Simulated Annealing algorithm with Exponential Temperature Schedule.

Configuration from Previous Knowledge

- We now have reasonable parameter values for our Simulated Annealing algorithm with Exponential Temperature Schedule.
- We have a rough impression about how far local optima under the unary operator are apart in terms of objective value (about 50).

Configuration from Previous Knowledge

- We now have reasonable parameter values for our Simulated Annealing algorithm with Exponential Temperature Schedule.
- We have a rough impression about how far local optima under the unary operator are apart in terms of objective value (about 50).
- We used this to obtain a reasonable start temperature $T_s = 20$.

Configuration from Previous Knowledge

- We now have reasonable parameter values for our Simulated Annealing algorithm with Exponential Temperature Schedule.
- We have a rough impression about how far local optima under the unary operator are apart in terms of objective value (about 50).
- We used this to obtain a reasonable start temperature $T_s = 20$.
- We can choose a reasonably small end temperature T_e.

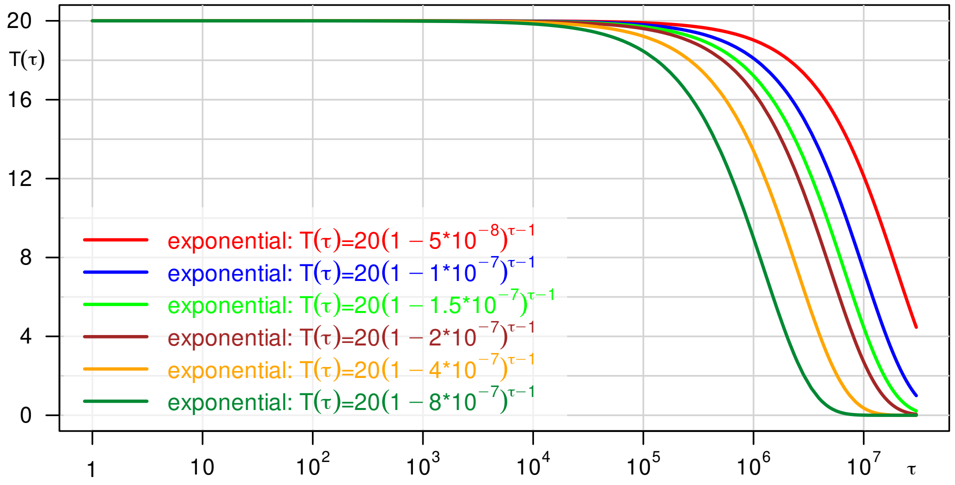
Configuration from Previous Knowledge

- We now have reasonable parameter values for our Simulated Annealing algorithm with Exponential Temperature Schedule.
- We have a rough impression about how far local optima under the unary operator are apart in terms of objective value (about 50).
- We used this to obtain a reasonable start temperature $T_s = 20$.
- We can choose a reasonably small end temperature T_e.
- We did this by setting $T_e = 0.1$ such that we would accept a solution which is $\Delta E = 1$ worse than the current solution about every $L = 16'384$ steps (which was the length until the hill climber would do a restart).

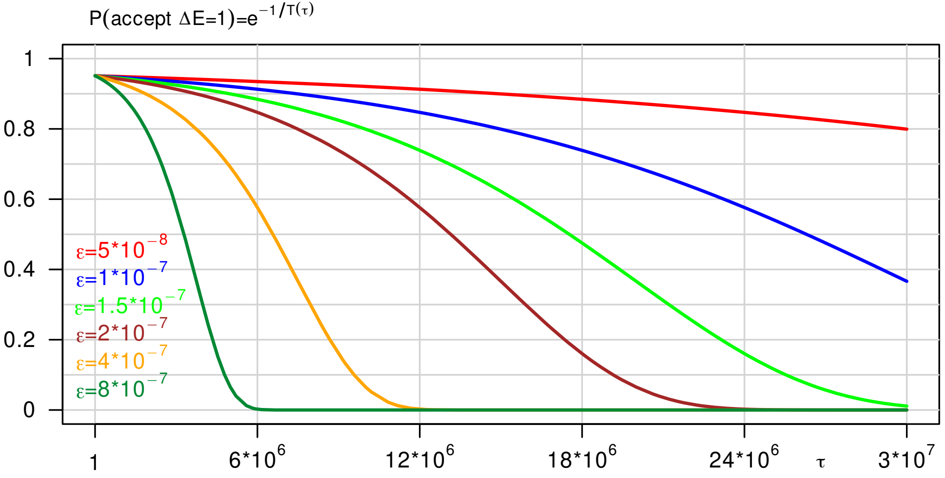
Configuration from Previous Knowledge

- We now have reasonable parameter values for our Simulated Annealing algorithm with Exponential Temperature Schedule.
- We have a rough impression about how far local optima under the unary operator are apart in terms of objective value (about 50).
- We used this to obtain a reasonable start temperature $T_s = 20$.
- We can choose a reasonably small end temperature T_e.
- We did this by setting $T_e = 0.1$ such that we would accept a solution which is $\Delta E = 1$ worse than the current solution about every $L = 16'384$ steps (which was the length until the hill climber would do a restart).
- Finally, by knowing that we can do about 30'000'000 FEs in total, we can set $\varepsilon \in \left[1 * 10^{-7}, 2 * 10^{-7}\right]$ such that T_e would be reached near the end of the runs.

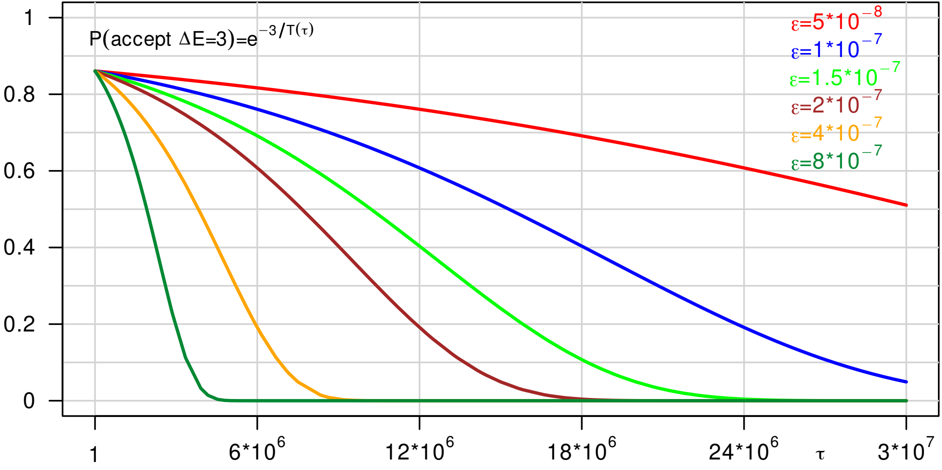
Behavior of the Configurations



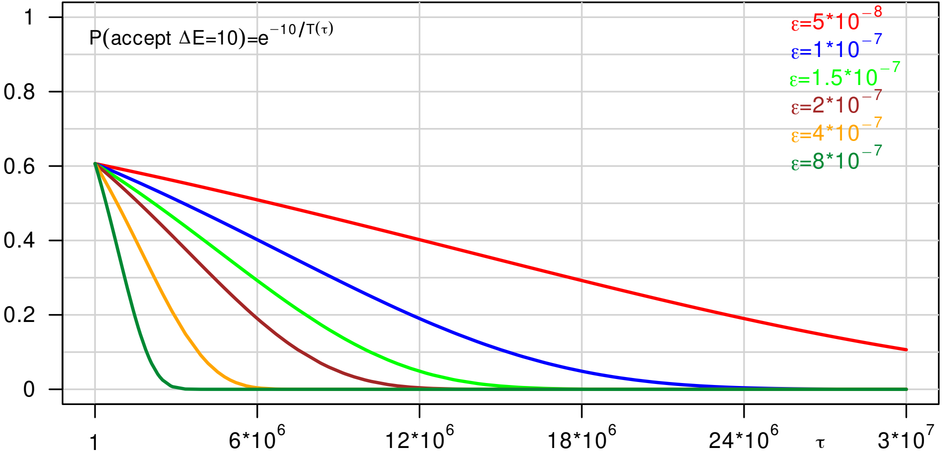
Behavior of the Configurations

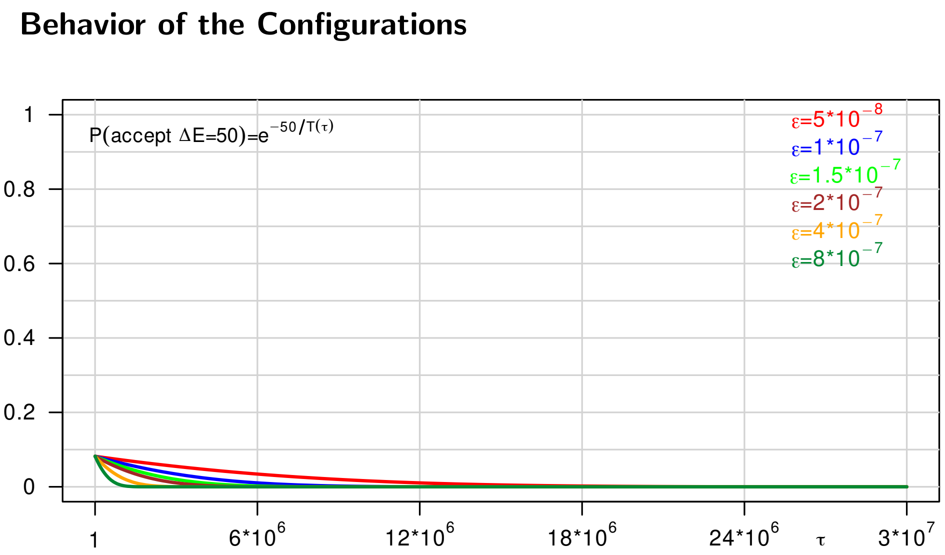


Behavior of the Configurations



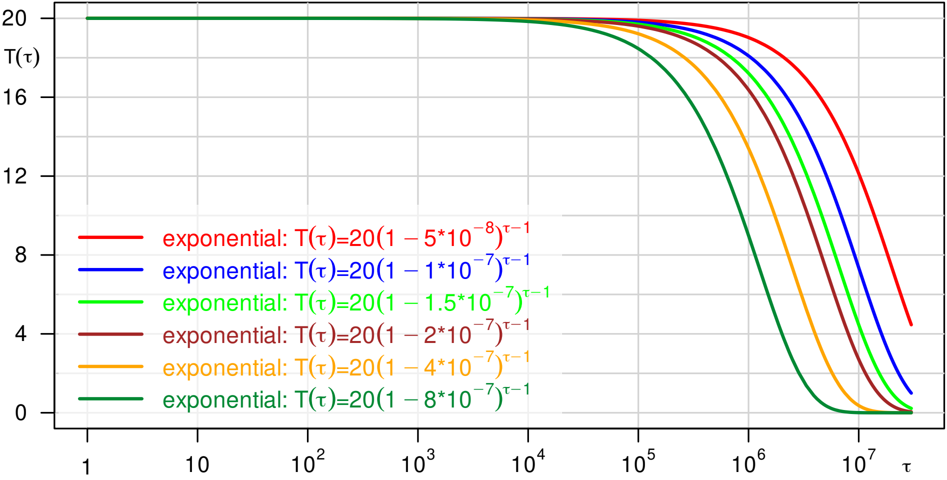
Behavior of the Configurations





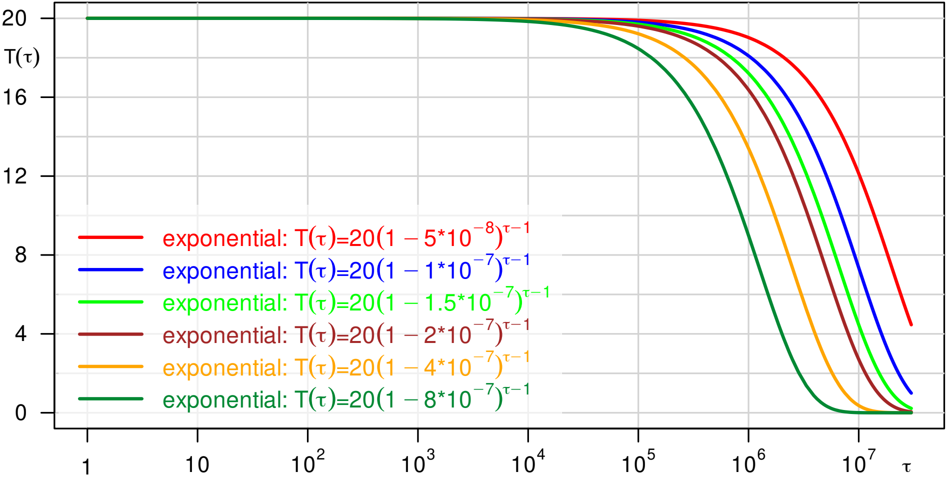
Behavior of the Configurations
P(accept ΔE=50)=e^{−50/T(τ)}
ε=5*10^−8
ε=1*10^−7
ε=1.5*10^−7
ε=2*10^−7
ε=4*10^−7
ε=8*10^−7
1
0.8
0.6
0.4
0.2
0
1
6*10^6
12*10^6
18*10^6
24*10^6
τ
3*10^7

Behavior of the Configurations



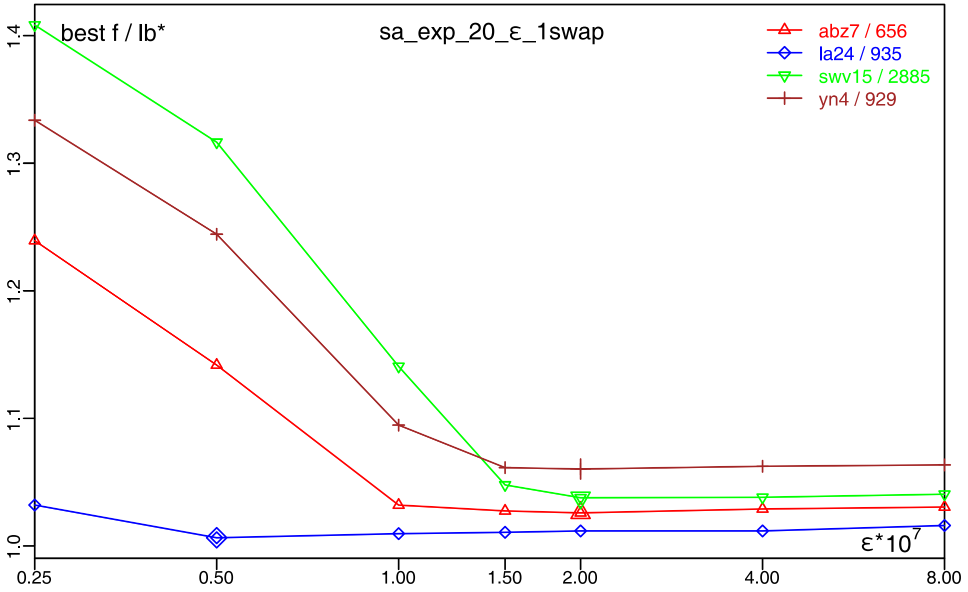
- Our very rough calculations gave us parameter settings for T_s and ε that produce these temperature- and probability curves.

Behavior of the Configurations

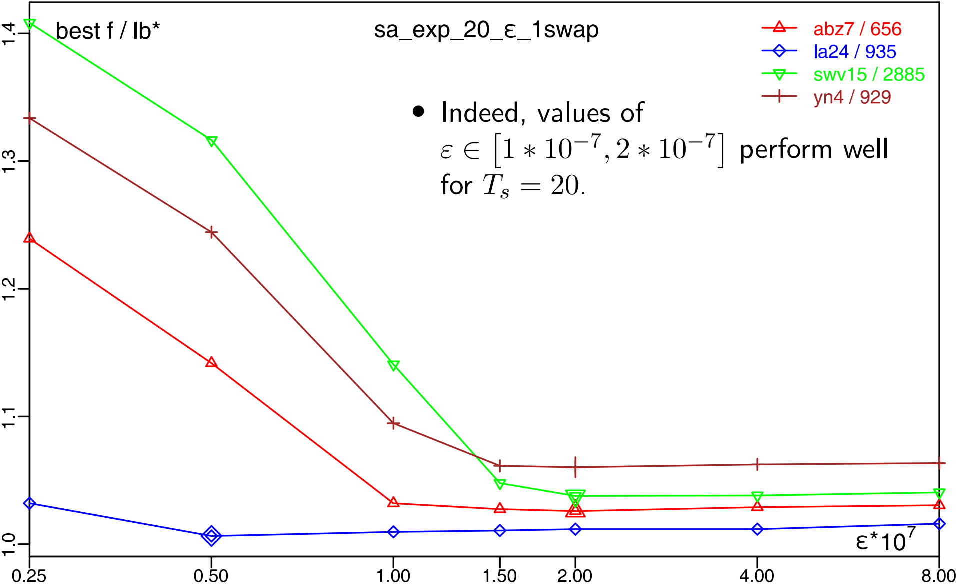


- Our very rough calculations gave us parameter settings for T_s and ε that produce these temperature- and probability curves.
- Whether these settings are actually any good must be studied now.

Relation of ε and Performance

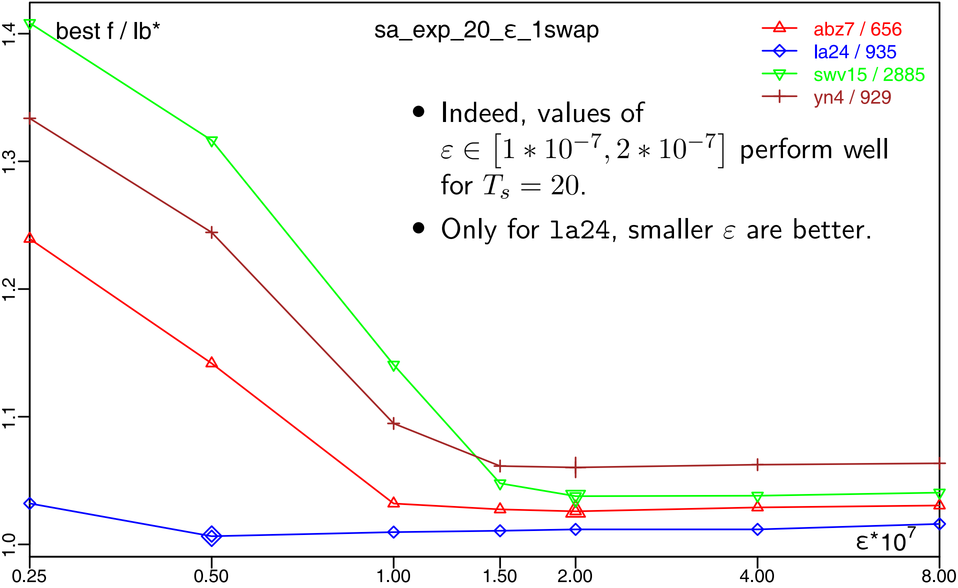


Relation of ε and Performance



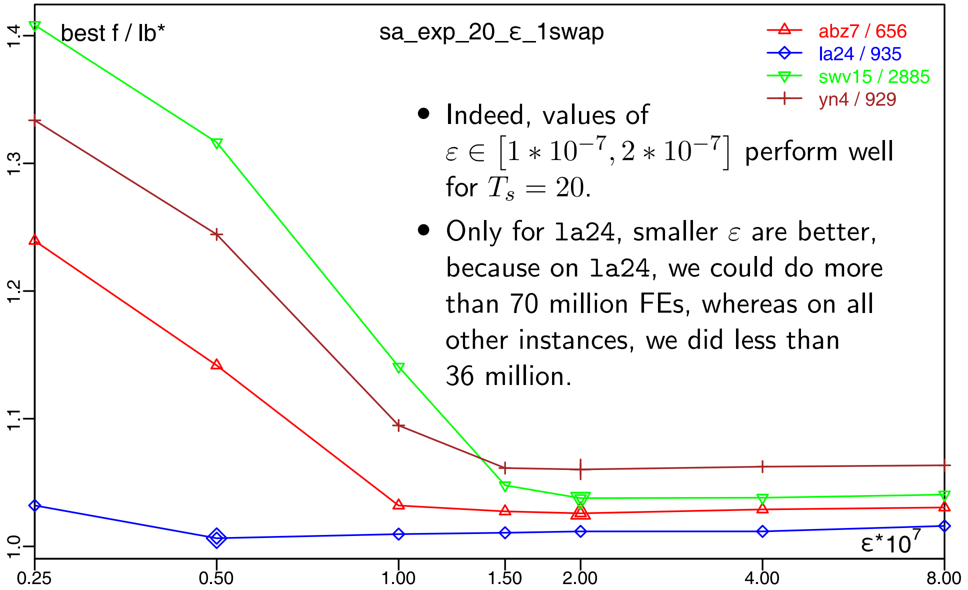
- Indeed, values of $\varepsilon \in \left[1 * 10^{-7}, 2 * 10^{-7}\right]$ perform well for $T_s = 20$.

Relation of ε and Performance



- Indeed, values of $\varepsilon \in \left[1 * 10^{-7}, 2 * 10^{-7}\right]$ perform well for $T_s = 20$.
- Only for `la24`, smaller ε are better.

Relation of ε and Performance



- Indeed, values of $\varepsilon \in \left[1 * 10^{-7}, 2 * 10^{-7}\right]$ perform well for $T_s = 20$.
- Only for `la24`, smaller ε are better, because on `la24`, we could do more than 70 million FEs, whereas on all other instances, we did less than 36 million.

Experiment and Analysis

So what do we get?

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

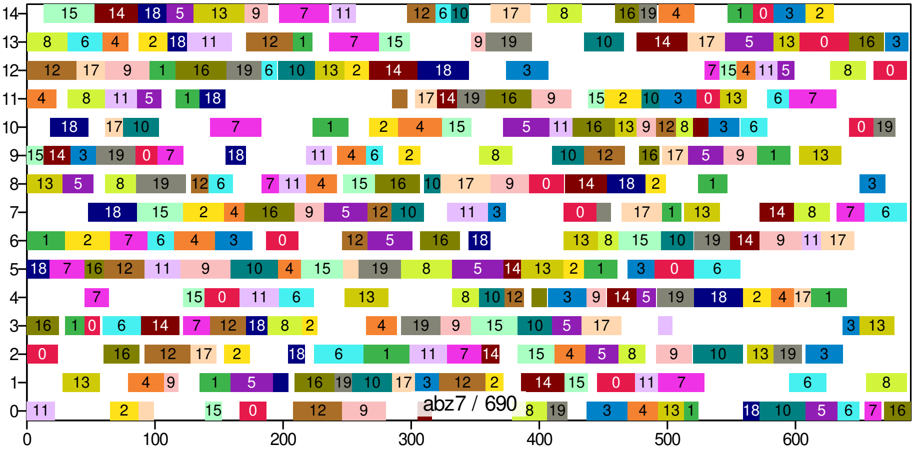
So what do we get?

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

		makespan				**last improvement**	
$\mathcal{I}$	algo	best	mean	med	sd	med(t)	med(FEs)
abz7	hcr_65536_nswap	712	731	732	6	96s	21'189'358
	eac_4_5%_nswap	672	690	690	9	68s	12'474'571
	sa_exp_20_2_1swap	**663**	**673**	**673**	**5**	112s	21'803'600
la24	hcr_65536_nswap	942	973	974	**8**	71s	31'466'420
	eac_4_5%_nswap	**935**	963	961	16	30s	9'175'579
	sa_exp_20_2_1swap	938	**949**	**946**	8	33s	12'358'941
swv15	hcr_65536_nswap	3740	3818	3826	35	89s	10'783'296
	eac_4_5%_nswap	3102	3220	3224	65	168s	18'245'534
	sa_exp_20_2_1swap	**2936**	**2994**	**2994**	**28**	157s	20'045'507
yn4	hcr_65536_nswap	1068	1109	1110	12	78s	18'756'636
	eac_4_5%_nswap	1000	1038	1037	18	118s	15'382'072
	sa_exp_20_2_1swap	**973**	**985**	**985**	**5**	130s	20'407'559

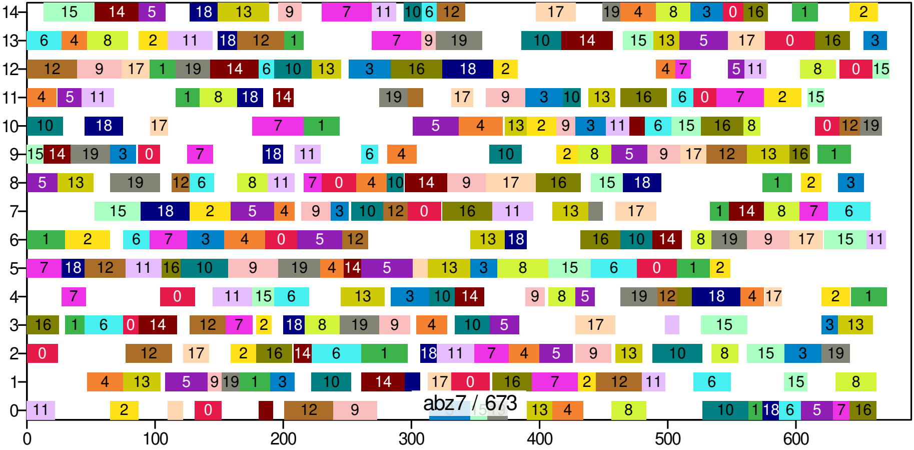
So what do we get?

eac_4_5%_nswap: median result of 3 min of the EA with clearing and $\mu = \lambda = 4$ with nswap unary operator and 5% sequence recombination



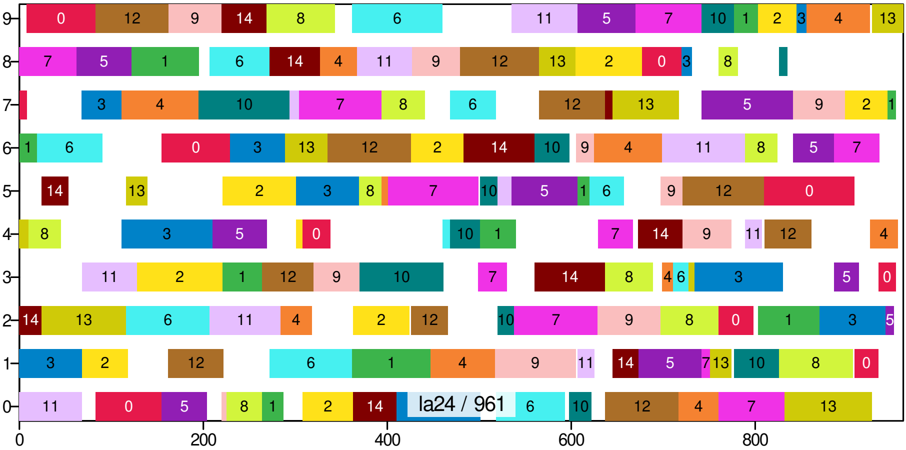
So what do we get?

sa_exp_20_2_1swap: median result of 3 min of Simulated Annealing with exponential schedule, $T_s = 20$, and $\varepsilon = 2 \cdot 10^{-7}$ and 1swap unary operator



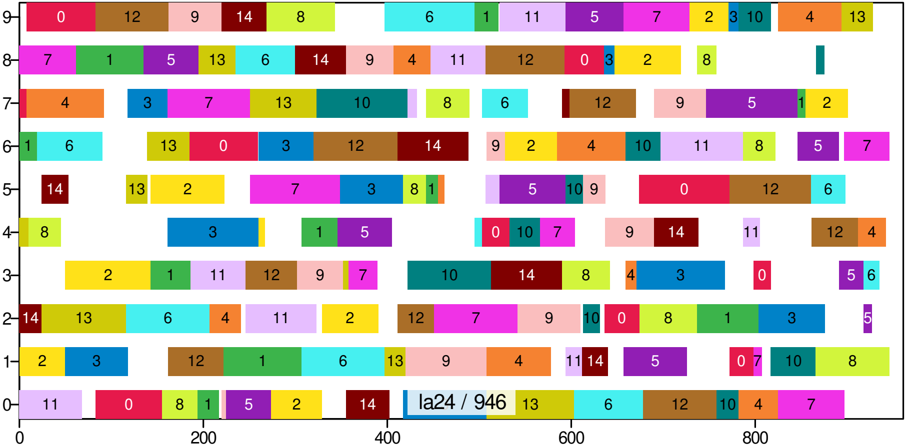
So what do we get?

`eac_4_5%_nswap`: median result of 3 min of the EA with clearing and $\mu = \lambda = 4$ with `nswap` unary operator and 5% `sequence` recombination



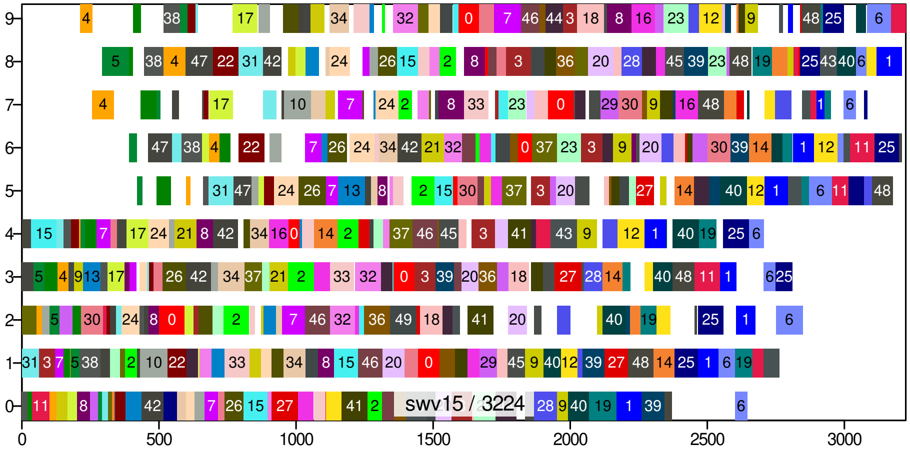
So what do we get?

`sa_exp_20_2_1swap`: median result of 3 min of Simulated Annealing with exponential schedule, $T_s = 20$, and $\varepsilon = 2 \cdot 10^{-7}$ and `1swap` unary operator



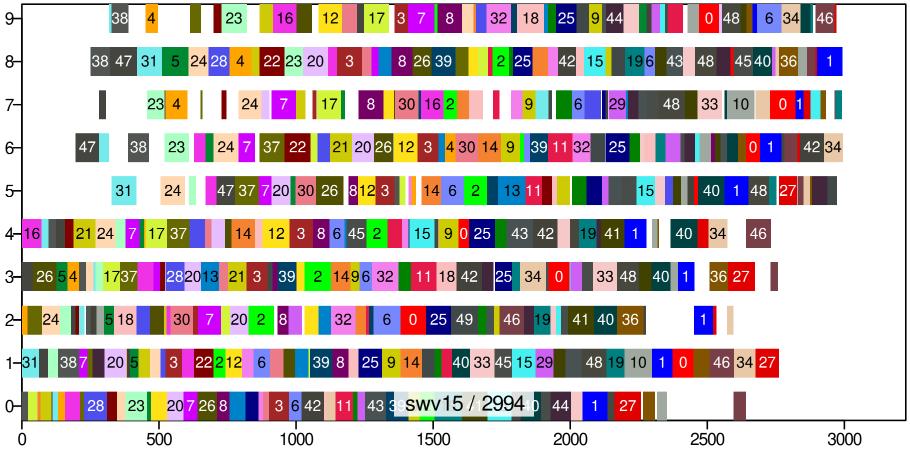
So what do we get?

`eac_4_5%_nswap`: median result of 3 min of the EA with clearing and $\mu = \lambda = 4$ with `nswap` unary operator and 5% sequence recombination



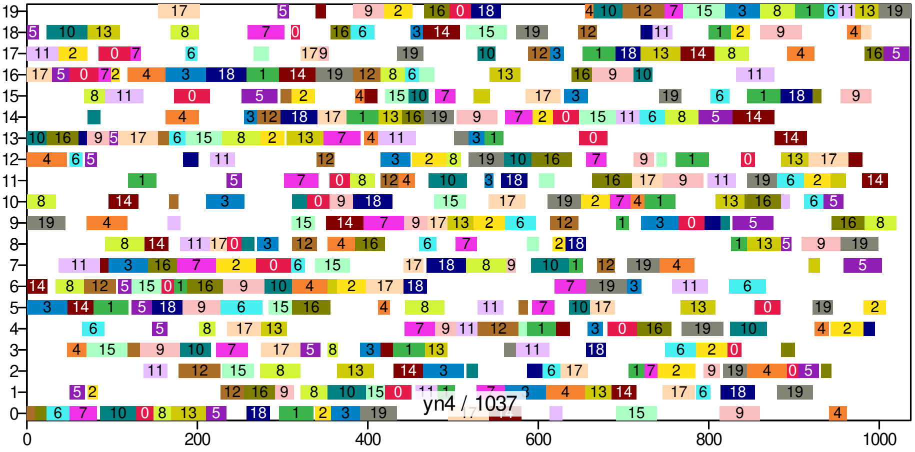
So what do we get?

sa_exp_20_2_1swap: median result of 3 min of Simulated Annealing with exponential schedule, $T_s = 20$, and $\varepsilon = 2 \cdot 10^{-7}$ and 1swap unary operator



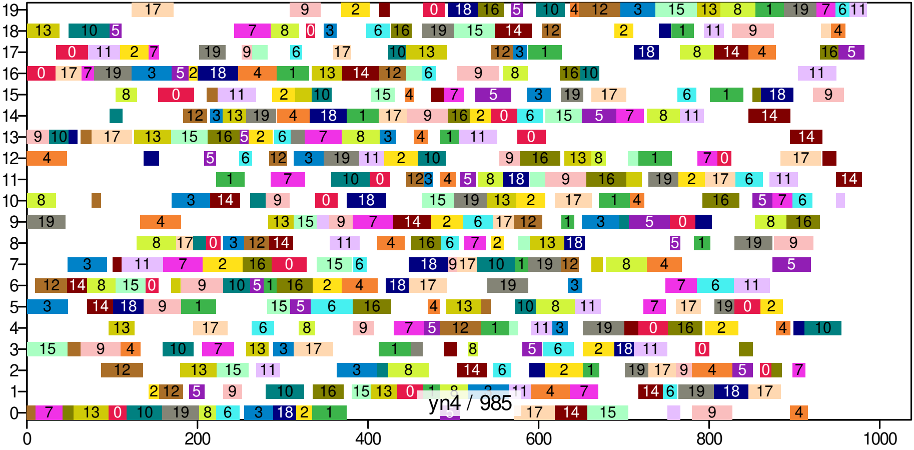
So what do we get?

`eac_4_5%_nswap`: median result of 3 min of the EA with clearing and $\mu = \lambda = 4$ with `nswap` unary operator and 5% `sequence` recombination



So what do we get?

sa_exp_20_2_1swap: median result of 3 min of Simulated Annealing with exponential schedule, $T_s = 20$, and $\varepsilon = 2 \cdot 10^{-7}$ and 1swap unary operator

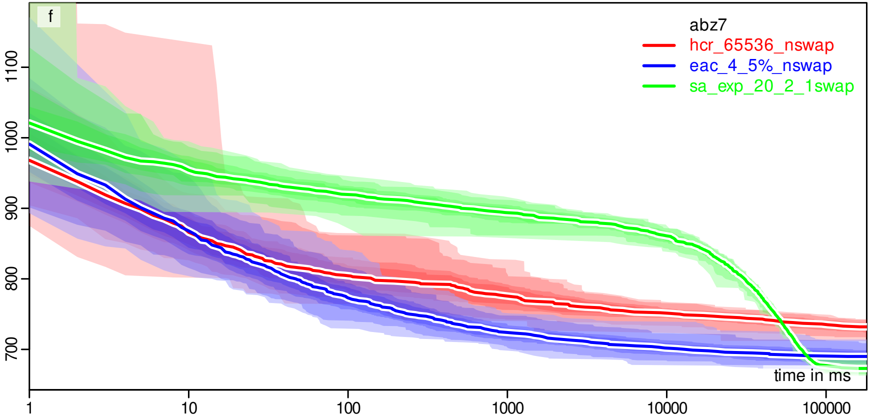


Progress over Time

What progress does the algorithm make over time?

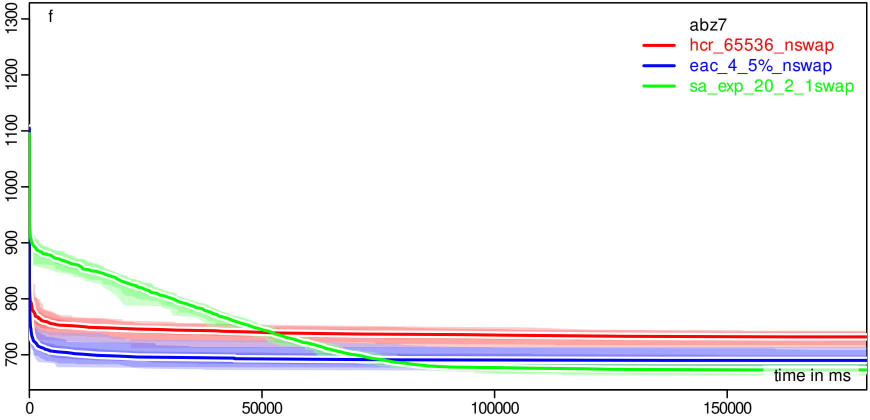
Progress over Time

What progress does the algorithm make over time?



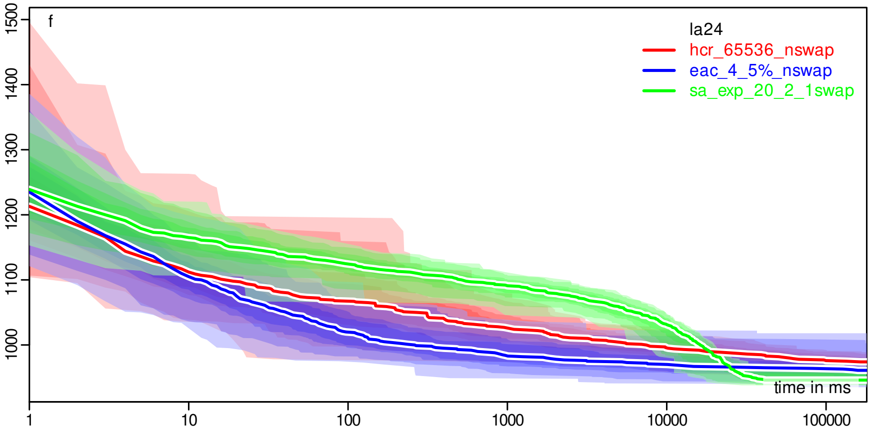
Progress over Time

What progress does the algorithm make over time?



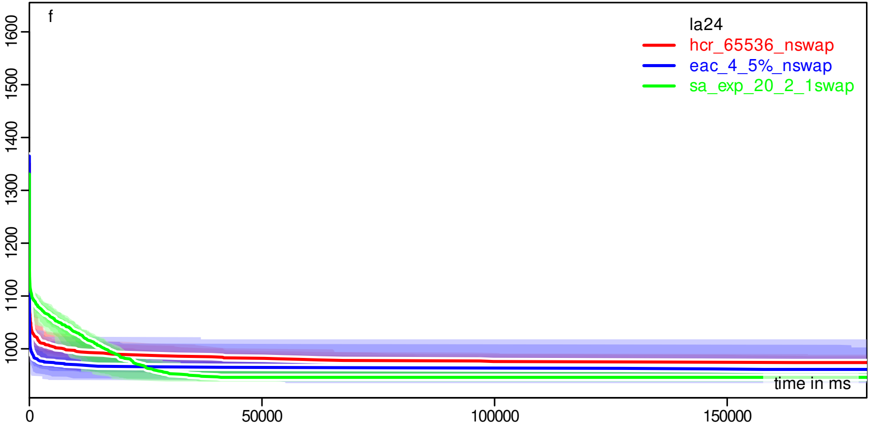
Progress over Time

What progress does the algorithm make over time?

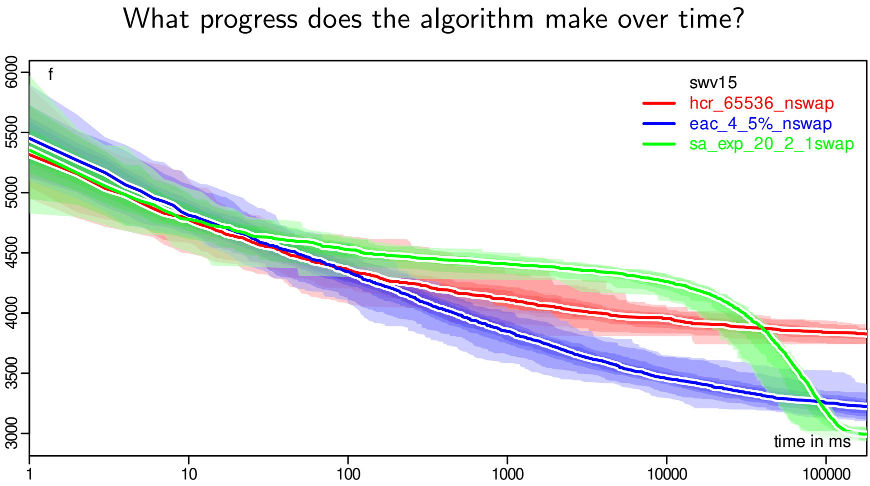


Progress over Time

What progress does the algorithm make over time?

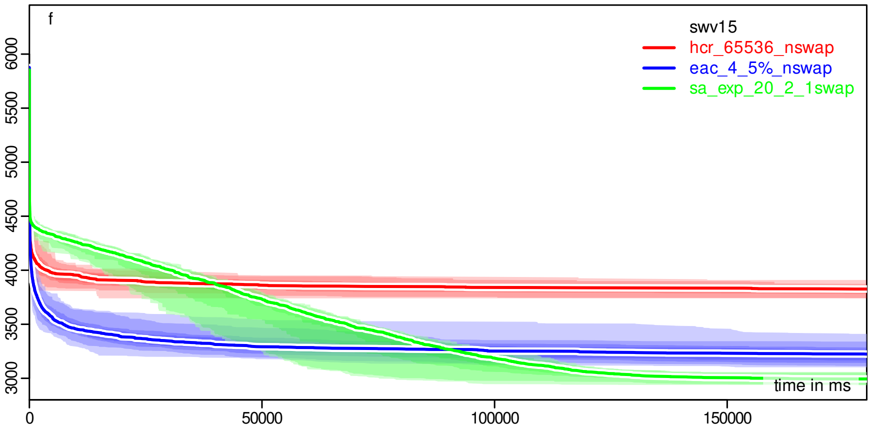


Progress over Time



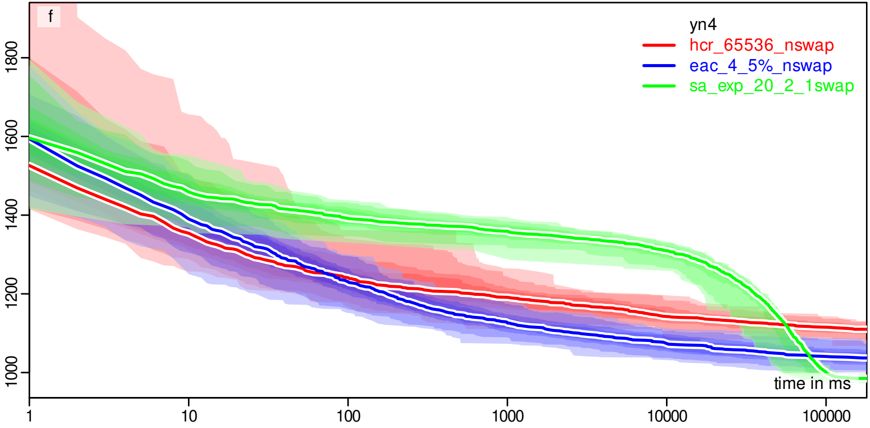
Progress over Time

What progress does the algorithm make over time?



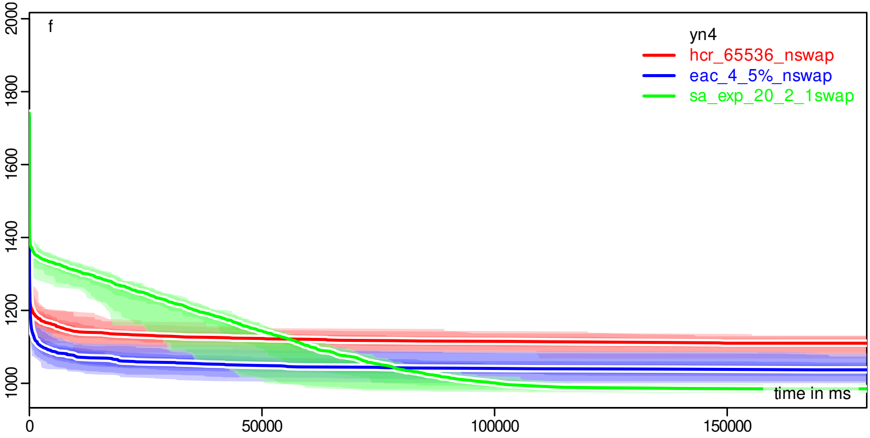
Progress over Time

What progress does the algorithm make over time?



Progress over Time

What progress does the algorithm make over time?



Simulated Annealing is better than the other algorithms and keeps improving longer.

Optimal Solutions for `la24`

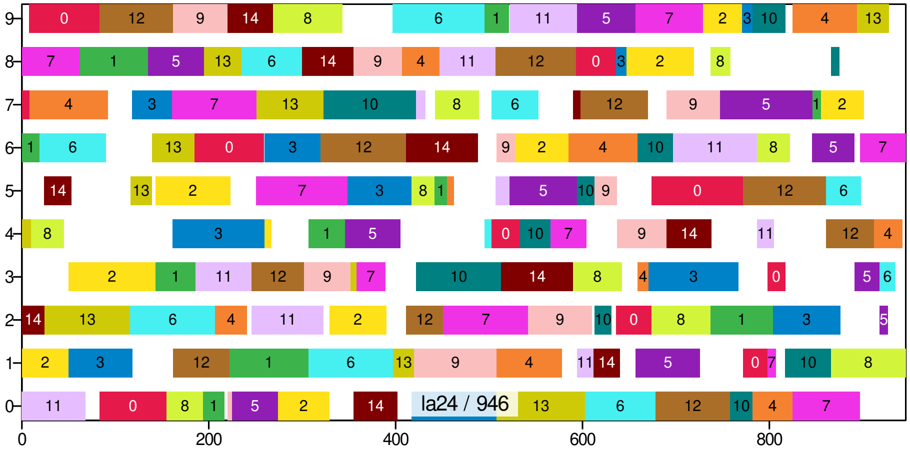
- Interestingly, the setups with $\varepsilon = 4 \cdot 10^{-7}$ and $\varepsilon = 8 \cdot 10^{-7}$, which we did not choose for our summary, each found one solution for `la24` with makespan 935.

Optimal Solutions for `la24`

- Interestingly, the setups with $\varepsilon = 4 \cdot 10^{-7}$ and $\varepsilon = 8 \cdot 10^{-7}$, which we did not choose for our summary, each found one solution for `la24` with makespan 935.
- Since we know that the lower bound for the makespan on `la24` is also 935[12] [13], we know that we found two globally optimal, best possible solutions!

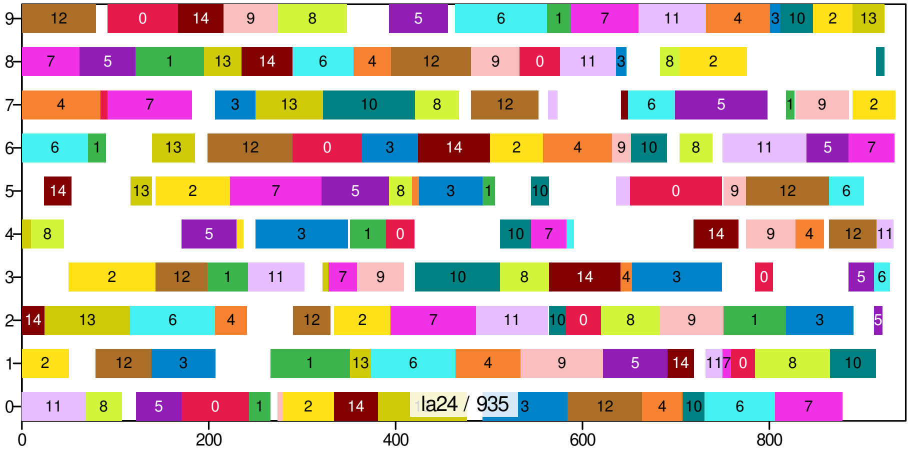
Optimal Solutions for `la24`

`sa_exp_20_2_1swap`: median result of 3 min of Simulated Annealing with exponential schedule, $T_s = 20$, and $\varepsilon = 2 \cdot 10^{-7}$ and `1swap` unary operator



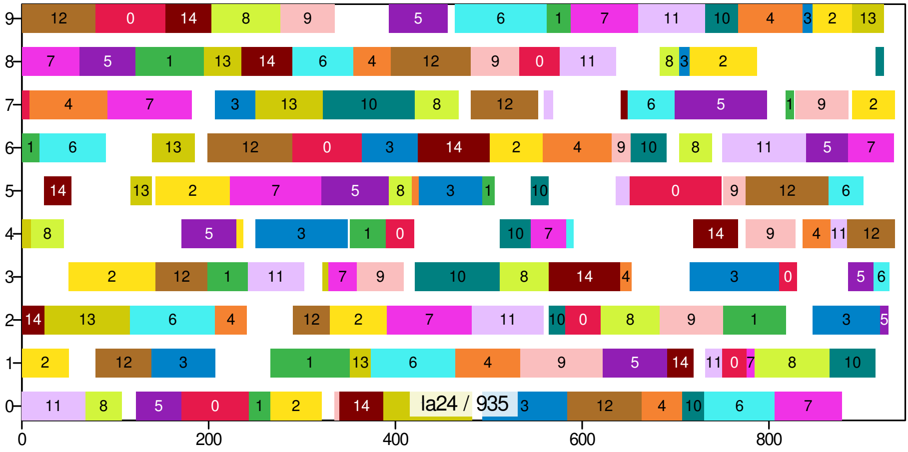
Optimal Solutions for `la24`

`sa_exp_20_4_1swap`: best result of 3 min of Simulated Annealing with exponential schedule, $T_s = 20$, and $\varepsilon = 4 \cdot 10^{-7}$ and `1swap` unary operator



Optimal Solutions for `la24`

`sa_exp_20_8_1swap`: best result of 3 min of Simulated Annealing with exponential schedule, $T_s = 20$, and $\varepsilon = 8 \cdot 10^{-7}$ and `1swap` unary operator



Summary

- Simulated Annealing outperformed all algorithms that we have tested before.

Summary

- Simulated Annealing outperformed all algorithms that we have tested before.
- We can also use knowledge to configure the algorithm more quickly.

Summary

- Simulated Annealing outperformed all algorithms that we have tested before.
- We can also use knowledge to configure the algorithm more quickly. For this, we need to know approximately:
  - approximately how many algorithm steps we can do within the computational budget.

Summary

- Simulated Annealing outperformed all algorithms that we have tested before.
- We can also use knowledge to configure the algorithm more quickly. For this, we need to know approximately:
  - approximately how many algorithm steps we can do within the computational budget and
  - what “a bit worse” means in terms of the objective function.

Summary

- Simulated Annealing outperformed all algorithms that we have tested before.
- We can also use knowledge to configure the algorithm more quickly. For this, we need to know approximately:
  - approximately how many algorithm steps we can do within the computational budget and
  - what "a bit worse" means in terms of the objective function.
  - We then can determine a starting temperature T_s and a parameter ϵ to tune the temperature schedule accordingly.

Summary

- Simulated Annealing outperformed all algorithms that we have tested before.
- We can also use knowledge to configure the algorithm more quickly. For this, we need to know approximately:
  - approximately how many algorithm steps we can do within the computational budget and
  - what “a bit worse” means in terms of the objective function.
  - We then can determine a starting temperature T_s and a parameter ϵ to tune the temperature schedule accordingly.
- Perspective

Summary

- Simulated Annealing outperformed all algorithms that we have tested before.
- We can also use knowledge to configure the algorithm more quickly. For this, we need to know approximately:
  - approximately how many algorithm steps we can do within the computational budget and
  - what "a bit worse" means in terms of the objective function.
  - We then can determine a starting temperature T_s and a parameter ϵ to tune the temperature schedule accordingly.
- Perspective:
  - An Evolutionary Algorithm allows us to pick a behavior in between a hill climber and a random sampling algorithm by choosing a small or large population size.

Summary

- Simulated Annealing outperformed all algorithms that we have tested before.
- We can also use knowledge to configure the algorithm more quickly. For this, we need to know approximately:
  - approximately how many algorithm steps we can do within the computational budget and
  - what "a bit worse" means in terms of the objective function.
  - We then can determine a starting temperature T_s and a parameter ϵ to tune the temperature schedule accordingly.
- Perspective:
  - An Evolutionary Algorithm allows us to pick a behavior in between a hill climber and a random sampling algorithm by choosing a small or large population size.
  - The Simulated Annealing algorithm allows for a smooth transition of a random search behavior towards a hill climbing behavior over time.

Summary

- Simulated Annealing outperformed all algorithms that we have tested before.
- We can also use knowledge to configure the algorithm more quickly. For this, we need to know approximately:
  - approximately how many algorithm steps we can do within the computational budget and
  - what "a bit worse" means in terms of the objective function.
  - We then can determine a starting temperature T_s and a parameter ϵ to tune the temperature schedule accordingly.
- Perspective:
  - An Evolutionary Algorithm allows us to pick a behavior in between a hill climber and a random sampling algorithm by choosing a small or large population size.
  - The Simulated Annealing algorithm allows for a smooth transition of a random search behavior towards a hill climbing behavior over time.
  - Compared to the hill climber with restarts, it offers a "softer" way to escape from local optima which sacrifices less solution information.

谢谢

Thank you

References I

1. Thomas Weise. *An Introduction to Optimization Algorithms*. Institute of Applied Optimization (IAO) [应用优化研究所] of the School of Artificial Intelligence and Big Data [人工智能与大数据学院] of Hefei University [合肥学院], Hefei [合肥市], Anhui [安徽省], China [中国], 2018–2020. URL http://thomasweise.github.io/aitoa/.
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. Scott Kirkpatrick, C. Daniel Gelatt, Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science Magazine*, 220 (4598):671–680, May 13, 1983. doi:10.1126/science.220.4598.671. URL http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.4175.
4. Vladimír Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, January 1985. doi:10.1007/BF00940812. URL http://mkweb.bcgsc.ca/papers/cerny-travelingsalesman.pdf.
5. Dean Jacobs, Jan Prins, Peter Siegel, and Kenneth Wilson. Monte carlo techniques in code optimization. *ACM SIGMICRO Newsletter*, 13(4):143–148, December 1982. Proceedings of the 15th Annual Workshop on Microprogramming (MICRO 15), October 5–7, 1982, Palo Alto, CA, USA, New York, NY, USA: ACM.
6. Martin Pincus. Letter to the editor – a monte carlo method for the approximate solution of certain types of constrained optimization problems. *Operations Research*, 18(6):1225–1228, November–December 1970. doi:10.1287/opre.18.6.1225.
7. James C. Spall. *Introduction to Stochastic Search and Optimization*, volume 6 of *Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization*. Wiley Interscience, Chichester, West Sussex, UK, April 2003. ISBN 0-471-33052-3. URL http://www.jhuapl.edu/ISSO/.
8. John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, Ann Arbor, MI, USA, 1975. ISBN 0-472-08460-7.
9. Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Computational Intelligence Library. Oxford University Press, Inc., New York, NY, USA, 1997. ISBN 0-7503-0392-1.
10. Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, Berlin/Heidelberg, 2nd edition, 2004. ISBN 3-540-22494-7.
11. David Edward Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989. ISBN 0-201-15767-5.
12. David Lee Applegate and William John Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, May 1991. doi:10.1287/ijoc.3.2.149. the JSSP instances used were generated in Bonn in 1986.
13. Jelke Jeroen van Hoorn. Job shop instances and solutions, 2015. URL http://jobshop.jjvh.nl.