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)
School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

应用优化研究所
人工智能与大数据学院
合肥学院
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 \texttt{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 \( n_{\text{swap}} \) 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 \texttt{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 $\texttt{nswap}$ 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 $\texttt{1swap}$ operator.
Idea

- A schedule which is a local optimum (under 1\textit{swap}) 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 \texttt{1\textasciitilde swap}) 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 \texttt{1\textasciitilde swap} unary search operator that we use.
Idea

• A schedule which is a local optimum (under \texttt{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 \texttt{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 1\text{swap}) 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 1\text{swap} 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 1\text{swap}) 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 1\text{swap} 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)\textsuperscript{3–6} is a local search which provides another approach to escape from local optima\textsuperscript{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)\textsuperscript{3–6} is a local search which provides another approach to escape from local optima\textsuperscript{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 \(\Delta E\) 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 $\Delta E$ 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)\textsuperscript{3–6} is a local search which provides another approach to escape from local optima\textsuperscript{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 $\Delta E$ 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)\textsuperscript{3–6} is a local search which provides another approach to escape from local optima\textsuperscript{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 $\Delta E$ 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 \(x \in X\) and that we have derived a new point \(x' \in X\) from it using the unary search operator.
• \(\Delta E\) 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.
• $\Delta E$ then be the difference between the objective value of $x'$ and $x$:

$$\Delta E = f(\gamma(x')) - f(\gamma(x))$$ (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.
• $\Delta E$ then be the difference between the objective value of $x'$ and $x$:

$$\Delta E = f(\gamma(x')) - f(\gamma(x))$$  \hspace{1cm} (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.
• \( \Delta E \) 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 \( x \in X \) and that we have derived a new point \( x' \in X \) from it using the unary search operator.
- \( \Delta E \) then be the difference between the objective value of \( x' \) and \( x \):

\[
\Delta E = f(\gamma(x')) - f(\gamma(x))
\]  

(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 \( x \in X \) and that we have derived a new point \( x' \in X \) from it using the unary search operator.
- \( \Delta E \) then be the difference between the objective value of \( x' \) and \( x \):

\[
\Delta E = f(\gamma(x')) - f(\gamma(x))
\]  

(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.
• $\Delta E$ then be the difference between the objective value of $x'$ and $x$:

$$\Delta E = f(\gamma(x')) - f(\gamma(x))$$ (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.
- $\Delta E$ then be the difference between the objective value of $x'$ and $x$:

\[
\Delta E = f(\gamma(x')) - f(\gamma(x))
\]  

(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

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  \hspace{1cm} (1)
Acceptance Probability

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  \hspace{1cm} (1)

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

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  \hspace{1cm} (1)

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

\[ P = \begin{cases} & \text{if } \Delta E < 0 \\ & \text{if } \Delta E \geq 0 \end{cases} \]  \hspace{1cm} (2)
Acceptance Probability

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  

(1)

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

\[ P = \begin{cases} 
1 
\end{cases} \]  

(2)
Acceptance Probability

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  

(1)

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

\[
P = \begin{cases} 
1 & \text{if } \Delta E \leq 0 
\end{cases}
\]  

(2)
**Acceptance Probability**

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  

(1)

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

\[
P = \begin{cases} 
1 & \text{if } \Delta E \leq 0 \\
 e^{-\frac{\Delta E}{T}} & \text{otherwise}
\end{cases}
\]  

(2)


**Acceptance Probability**

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  

(1)

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

\[
P = \begin{cases} 
1 & \text{if } \Delta E \leq 0 \\
\frac{e^{-\frac{\Delta E}{T}}}{T} & \text{if } \Delta E > 0 \land T > 0
\end{cases}
\]  

(2)
Acceptance Probability

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \quad (1) \]

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

\[
P = \begin{cases} 
1 & \text{if } \Delta E \leq 0 \\
\frac{1}{e^{\frac{\Delta E}{T}}} & \text{if } \Delta E > 0 \wedge T > 0 \\
0 & \text{if } \Delta E > 0 \wedge T \leq 0 
\end{cases} \quad (2)
\]
Acceptance Probability

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \] (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} \] (2)
Acceptance Probability

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  

(1)

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

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

(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

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  

(1)

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

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

(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

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  

(1)

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

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

(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

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  

(1)

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

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

(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 \( \Delta E \) is.
Acceptance Probability

\[ \Delta E = f(\gamma(x')) - f(\gamma(x)) \]  \hspace{1cm} (1)

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

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

- 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 \( \Delta E \) is and
  2. gets smaller the smaller the so-called “temperature” \( T \geq 0 \) is.
Ingredient: Temperature Schedule
Temperature Schedule

\[ P = \begin{cases} 
1 & \text{if } \Delta E \leq 0 \\
e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \land T > 0 \\
0 & \text{otherwise } (\Delta E > 0 \land T = 0) 
\end{cases} \]
Temperature Schedule

\[ P = \begin{cases} 
1 & \text{if } \Delta E \leq 0 \\
\exp\left(-\frac{\Delta E}{T} \right) & \text{if } \Delta E > 0 \land T > 0 \\
0 & \text{otherwise (} \Delta E > 0 \land T = 0) 
\end{cases} \]  

(2)

• What about this temperature \( T \)?
Temperature Schedule

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

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

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

- What about this temperature \( T \)?
- The temperature is defined to decrease and approaches zero with a rising number \( \tau \) of performed objective function evaluations.
- The optimization process is initially “hot” and \( T \) is high.
Temperature Schedule

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

(2)

• What about this temperature \(T\)?
• The temperature is defined to decrease and approaches zero with a rising number \(\tau\) 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

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

- What about this temperature \( T \)?
- The temperature is defined to decrease and approaches zero with a rising number \( \tau \) 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

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

- What about this temperature \( T \)?
- The temperature is defined to decrease and approaches zero with a rising number \( \tau \) 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

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

- What about this temperature \( T \)?
- The temperature is defined to decrease and approaches zero with a rising number \( \tau \) 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

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

(2)

• What about this temperature \( T(\tau) \)?
• The temperature is defined to decrease and approaches zero with a rising number \( \tau \) 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

\[ P = \begin{cases} 
1 & \text{if } \Delta E \leq 0 \\
\exp\left(-\frac{\Delta E}{T(\tau)}\right) & \text{if } \Delta E > 0 \land T(\tau) > 0 \\
0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0) 
\end{cases} \] (2)
Conditions for Temperature Schedule

\[ P = \begin{cases} 
1 & \text{if } \Delta E \leq 0 \\
\frac{\Delta E}{T(\tau)} & \text{if } \Delta E > 0 \land T(\tau) > 0 \\
0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0) 
\end{cases} \] (2)

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

\[ P = \begin{cases} 
1 & \text{if } \Delta E \leq 0 \\
\exp\left(-\frac{\Delta E}{T(\tau)}\right) & \text{if } \Delta E > 0 \land T(\tau) > 0 \\
0 & \text{otherwise (} \Delta E > 0 \land T(\tau) = 0) 
\end{cases} \]  

(2)

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

\[ P = \begin{cases} 
1 & \text{if } \Delta E \leq 0 \\
\frac{\Delta E}{T(\tau)} & \text{if } \Delta E > 0 \land T(\tau) > 0 \\
0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0) 
\end{cases} \] (2)

- The temperature \( T(\tau) \) is defined to decrease and approaches zero with a rising number \( \tau \) 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

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

• The temperature \( T(\tau) \) is defined to decrease and approaches zero with a rising number \( \tau \) 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.
package aitoa.algorithms;

public abstract class TemperatureSchedule {
// unnecessary things omitted here
    public final double startTemperature; // \( T_s \)

    public abstract double temperature(long tau); // \( T(\tau) \)
}

Base Class for Implementing Temperature Schedules
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.

$$T(\tau) = T_s \ast (1 - \varepsilon)^{\tau - 1}$$ (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.

$$T(\tau) = T_s \times (1 - \varepsilon)^{\tau-1}$$

- Higher values of $\varepsilon$ lead to a faster temperature decline.
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 $T_s$.

$$T(\tau) = \frac{T_s}{\ln(\varepsilon(\tau - 1) + e)}$$  \hfill (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 $T_s$.

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

• Larger values of $\varepsilon$ again lead to a faster temperature decline.
package aitoa.algorithms;

public class Logarithmic extends TemperatureSchedule {
    // unnecessary things omitted here
    public final double epsilon; // \equiv \varepsilon

    public double temperature(long tau) {
        // \( T(\tau) = \frac{T_s}{\ln(\varepsilon(\tau-1)+e)} \)
        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\textsuperscript{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\textsuperscript{2,8–11}.
• By using the population size parameters $\mu$ and $\lambda$, 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\textsuperscript{2} \textsuperscript{8–11}.
• By using the population size parameters $\mu$ and $\lambda$, 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\textsuperscript{2 8–11}.
• By using the population size parameters $\mu$ and $\lambda$, 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\textsuperscript{2 8–11}.
• By using the population size parameters $\mu$ and $\lambda$, 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!
Why do we have such a strange thing like a temperature schedule?
Let’s think back again about Evolutionary Algorithms\textsuperscript{2} 8–11.
By using the population size parameters $\mu$ and $\lambda$, 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 \(\mu\) and \(\lambda\), 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\textsuperscript{2 8–11}.
• By using the population size parameters \( \mu \) and \( \lambda \), 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\textsuperscript{2 8–11}.
• By using the population size parameters $\mu$ and $\lambda$, 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 \(\mu\) and \(\lambda\), we can tune the behavior of an EA between random sampling \((\mu \rightarrow \infty\) or \(\lambda \rightarrow \infty)\) 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 \(\mu\) and \(\lambda\), we can tune the behavior of an EA between random sampling \((\mu \rightarrow \infty \text{ or } \lambda \rightarrow \infty)\) 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

```java
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

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

        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

        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

```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
        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

```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() < 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) || (random.nextDouble() < Math.exp((fCur - fNew) / this.schedule.temperature(tau))))) {
                fCur = fNew; // update current objective value
                process.getSearchSpace().copy(xNew, xCur); // copy xNew to xCur
            } else { // 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 $\varepsilon$
• Our algorithm has four parameters:
  1. the start temperature $T_s$,
  2. the parameter $\varepsilon$,
  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 $\varepsilon$,
  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).
Our algorithm has four parameters:
1. the start temperature $T_s$,
2. the parameter $\varepsilon$,
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 $\varepsilon$,
  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 $\varepsilon$ to be configured.
Configuring the Algorithm

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

<table>
<thead>
<tr>
<th>$I$</th>
<th>med(total FEs)</th>
<th>sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>abz7</td>
<td>35'648'639</td>
<td>28</td>
</tr>
<tr>
<td>la24</td>
<td>70'952'285</td>
<td>56</td>
</tr>
<tr>
<td>swv15</td>
<td>21'662'286</td>
<td>137</td>
</tr>
<tr>
<td>yn4</td>
<td>27'090'511</td>
<td>48</td>
</tr>
<tr>
<td>median</td>
<td>31'369'575</td>
<td>52</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>( \mathcal{I} )</th>
<th>\text{med(total FEs)}</th>
<th>sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>abz7</td>
<td>35'648'639</td>
<td>28</td>
</tr>
<tr>
<td>la24</td>
<td>70'952'285</td>
<td>56</td>
</tr>
<tr>
<td>swv15</td>
<td>21'662'286</td>
<td>137</td>
</tr>
<tr>
<td>yn4</td>
<td>27'090'511</td>
<td>48</td>
</tr>
<tr>
<td>median</td>
<td>31'369'575</td>
<td>52</td>
</tr>
</tbody>
</table>

- \text{hc\_1swap} performs 30 million FEs (within the three minute budget) in \text{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.

<table>
<thead>
<tr>
<th></th>
<th>med(total FEs)</th>
<th>sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>abz7</td>
<td>35'648'639</td>
<td>28</td>
</tr>
<tr>
<td>la24</td>
<td>70'952'285</td>
<td>56</td>
</tr>
<tr>
<td>swv15</td>
<td>21'662'286</td>
<td>137</td>
</tr>
<tr>
<td>yn4</td>
<td>27'090'511</td>
<td>48</td>
</tr>
<tr>
<td>median</td>
<td><strong>31’369’575</strong></td>
<td>52</td>
</tr>
</tbody>
</table>

• 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.

<table>
<thead>
<tr>
<th>I</th>
<th>med(total FEs)</th>
<th>sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>abz7</td>
<td>35'648'639</td>
<td>28</td>
</tr>
<tr>
<td>la24</td>
<td>70'952'285</td>
<td>56</td>
</tr>
<tr>
<td>swv15</td>
<td>21'662'286</td>
<td>137</td>
</tr>
<tr>
<td>yn4</td>
<td>27'090'511</td>
<td>48</td>
</tr>
<tr>
<td>median</td>
<td><strong>31'369'575</strong></td>
<td>52</td>
</tr>
</tbody>
</table>

- `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

- $abz7$
- $rs$
- $hc\_1swap$

Time in ms
End Result Standard Deviation

![Graph showing the relationship between time in ms and standard deviation for different conditions: swv15, rs, and hc_1swap. The x-axis represents time in milliseconds (1 to 100,000), and the y-axis represents standard deviation from 4000 to 7000. Each condition has a distinct line with shaded confidence intervals.](image-url)
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 \( \text{hc}_1\text{swap} \) of 50 tells us something about the local optima.
- We know that \( \text{hc}_1\text{swap} \) 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.
• 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 $T_s$. 
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 $T_s$

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

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

$$P = e^{-\frac{\Delta E}{T(\tau)}}$$  \hspace{1cm} (5)
• 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 $T_s$:

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

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

$$0.1 = e^{-\frac{\Delta E}{T(\tau)}}$$  \hspace{1cm} (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 $T_s$:

\[
0.1 = e^{-\frac{\Delta E}{T(\tau)}}
\]  

(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 $T_s$:

\[
0.1 = e^{-\frac{50}{T(\tau)}}
\]  

(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 $T_s$:

$$0.1 = e^{-\frac{50}{T(\tau)}}$$  (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 \( T_s \):

\[
0.1 = e^{-\frac{50}{T_s}}
\]  
(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 $T_s$:

\[
0.1 = e^{-\frac{50}{T_s}}
\]  \hspace{1cm} (5)
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 $T_s$:

$$\ln 0.1 = -\frac{50}{T_s}$$ (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 $T_s$:

$$\ln 0.1 = -\frac{50}{T_s} \quad (5)$$
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 $T_s$:

$$T_s = -\frac{50}{\ln 0.1}$$ (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 $T_s$:

$$T_s = -\frac{50}{\ln 0.1}$$ (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 $T_s$:

$$T_s \approx 21.714724095$$ (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 $T_s$:

$$T_s \approx 20$$ (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 $T_s$:

$$T_s \approx 20$$  \hspace{1cm} (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 $\tau$ 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 $\tau$ 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 $\tau$ 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 $\tau$ 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 $\tau$ 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 $\tau$ 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 \( \tau \) of steps and the actual end temperature \( T_e \) should probably be slightly above 0.
- Let us remember back our \( hcr_L_1\text{swap} \) algorithm, i.e., the 1\text{swap} 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 \( hcr_16384_1\text{swap} \).
- 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.

$$P = \begin{cases} 
1 & \text{if } \Delta E \leq 0 \\
\frac{\Delta E}{T(\tau)} & \text{if } \Delta E > 0 \land T(\tau) > 0 \\
0 & \text{otherwise } (\Delta E > 0 \land T(\tau) = 0)
\end{cases}$$  \hspace{1cm} (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.

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

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

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

(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.

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

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

(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.

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

(6)
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.

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

(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.

\[
\frac{1}{16'384} = e^{-\frac{\Delta E}{T_e}}
\]  

(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.

$$\frac{1}{16'384} = e^{-\frac{1}{T_e}}$$ (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.

$$\frac{1}{16'384} = e^{-\frac{1}{T_e}}$$ (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.

\[
\ln \frac{1}{16'384} = - \frac{1}{T_e} \quad (6)
\]
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.

$$\ln \frac{1}{16'384} = - \frac{1}{T_e} \quad (6)$$
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.

$$T_e = -\frac{1}{\ln \frac{1}{16\,384}}$$  \hspace{1cm} (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.

$$T_e = - \frac{1}{\ln \frac{1}{16'384}}$$ (6)
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.

$$T_e \approx 0.103049646$$ (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 $P_e = 1/L = 1/167384$ at the end of our Simulated Annealing runs.

$$T_e \approx 0.1$$
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.

$$T_e \approx 0.1$$ (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 $\varepsilon$ parameter.
Epsilon from End Temperature and Iteration

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

$$T(\tau) = T_s \ast (1 - \varepsilon)^{\tau-1}$$
Epsilon from End Temperature and Iteration

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

\[
T(\tau) = T_s \times (1 - \varepsilon)^{\tau-1}
\]
Epsilon from End Temperature and Iteration

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

$$T(\tau) = T_s \times (1 - \varepsilon)^{\tau - 1} \quad (3)$$
We now want to find a good setting for the $\varepsilon$ parameter.

This parameter plays a role in the exponential temperature schedule.

It relates the temperature $T(\tau)$ at a given iteration $\tau$ to the iteration index $\tau$.

In order to compute a rough guess for $\varepsilon$, we thus need a value for $\tau$ 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 $T_e$.

$$T(\tau) = T_s \times (1 - \varepsilon)^{\tau-1}$$

(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 is we want to solve Equation 3 for $\varepsilon$ is the iteration index $\tau$ 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 $\varepsilon$ is the iteration index $\tau$ 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 is we want to solve Equation 3 for $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$T(\tau) = T_s \ast (1 - \varepsilon)^{\tau-1}$$  \hspace{1cm} (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 $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

\[
T(\tau) = T_s \ast (1 - \varepsilon)^{\tau-1}
\] (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 $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$T_e = T_s \ast (1 - \varepsilon)^{\tau-1}$$  \hspace{1cm} (7)
Epsilon from End Temperature and Iteration

- We have a start temperature $T_s$ and an end temperature $T_e$.
- What we need is we want to solve Equation 3 for $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

\[
0.1 = T_s \times (1 - \varepsilon)^{\tau - 1}
\] (7)
Epsilon from End Temperature and Iteration

- We have a start temperature $T_s$ and an end temperature $T_e$.
- What we need is we want to solve Equation 3 for $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = T_s \ast (1 - \varepsilon)^{\tau-1} \quad (7)$$
• We have a start temperature $T_s$ and an end temperature $T_e$.
• What we need it we want to solve Equation 3 for $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = T_s * (1 - \varepsilon)^{30'000'000} - 1$$ (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 $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = T_s \times (1 - \varepsilon)^{30'000'000-1}$$  (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 \( \varepsilon \) is the iteration index \( \tau \) 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 \( \tau \) is the time when we can expect the runs to end: \( T_e = T(30’000’000) \) and \( \tau = 30’000’000 \).

\[
0.1 = T_s \times (1 - \varepsilon)^{29'999'999}
\] (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 $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = T_s \times (1 - \varepsilon)^{29'999'999}$$ (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 $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = 20 \times (1 - \varepsilon)^{29'999'999} \quad (7)$$
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 $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.1 = 20 \times (1 - \varepsilon)^{29'999'999} \quad (7)$$
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 $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

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

(7)
Epsilon from End Temperature and Iteration

- We have a start temperature $T_s$ and an end temperature $T_e$.
- What we need is we want to solve Equation 3 for $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.005 = (1 - \varepsilon)^{29'999'999} \quad (7)$$
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 $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.005 = (1 - \varepsilon)^{29'999'999} \quad (7)$$
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 $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$0.005^{1/29'999'999} = 1 - \varepsilon$$  (7)
• We have a start temperature $T_s$ and an end temperature $T_e$.

• What we need it we want to solve Equation 3 for $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

\[ 0.999999823389 \approx 1 - \varepsilon \] (7)
Epsilon from End Temperature and Iteration

- We have a start temperature $T_s$ and an end temperature $T_e$.
- What we need is to solve Equation 3 for $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$\varepsilon \approx 1 - 0.9999999823389$$ (7)
Epsilon from End Temperature and Iteration

- We have a start temperature $T_s$ and an end temperature $T_e$.
- What we need is we want to solve Equation 3 for $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$\varepsilon \approx 1 - 0.9999999823389$$ (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 $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$\varepsilon \approx 0.000\,000\,176\,610\,569$$ (7)
Epsilon from End Temperature and Iteration

- We have a start temperature $T_s$ and an end temperature $T_e$.
- What we need is we want to solve Equation 3 for $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30'000'000)$ and $\tau = 30'000'000$.

$$\varepsilon \approx 1.776 \times 10^{-7}$$  \hspace{1cm} (7)
Epsilon from End Temperature and Iteration

- We have a start temperature \( T_s \) and an end temperature \( T_e \).
- What we need is we want to solve Equation 3 for \( \varepsilon \) is the iteration index \( \tau \) 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 \( \tau \) is the time when we can expect the runs to end: \( T_e = T(30’000’000) \) and \( \tau = 30’000’000 \).

\[
\varepsilon \in \left[ 1 \ast 10^{-7}, 2 \ast 10^{-7} \right] 
\] (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 $\varepsilon$ is the iteration index $\tau$ 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 $\tau$ is the time when we can expect the runs to end: $T_e = T(30’000’000)$ and $\tau = 30’000’000$.

$$\varepsilon \in \left[1 \times 10^{-7}, 2 \times 10^{-7}\right]$$ (7)

• Values of $\varepsilon$ 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 $T_e$. 

• 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).
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 $\varepsilon \in [1 \times 10^{-7}, 2 \times 10^{-7}]$ such that $T_e$ would be reached near the end of the runs.
Behavior of the Configurations

exponential: \( T(\tau) = 20(1 - 5\times10^{-8})\tau^{-1} \)

exponential: \( T(\tau) = 20(1 - 1\times10^{-7})\tau^{-1} \)

exponential: \( T(\tau) = 20(1 - 1.5\times10^{-7})\tau^{-1} \)

exponential: \( T(\tau) = 20(1 - 2\times10^{-7})\tau^{-1} \)

exponential: \( T(\tau) = 20(1 - 4\times10^{-7})\tau^{-1} \)

exponential: \( T(\tau) = 20(1 - 8\times10^{-7})\tau^{-1} \)
Behavior of the Configurations

$P(\text{accept } \Delta E=1) = e^{-1/T(\tau)}$

- $\epsilon = 5 \times 10^{-8}$
- $\epsilon = 1 \times 10^{-7}$
- $\epsilon = 1.5 \times 10^{-7}$
- $\epsilon = 2 \times 10^{-7}$
- $\epsilon = 4 \times 10^{-7}$
- $\epsilon = 8 \times 10^{-7}$

$\tau$ ranges from $1$ to $3 \times 10^7$. The plot shows the probability of accepting $\Delta E = 1$ as a function of $\tau$. Each line corresponds to a different value of $\epsilon$.
Behavior of the Configurations

\[ P(\text{accept } \Delta E=3) = e^{-3/T(\tau)} \]

\[ \epsilon = 5 \times 10^{-8}, \quad \epsilon = 1 \times 10^{-7}, \quad \epsilon = 1.5 \times 10^{-7}, \quad \epsilon = 2 \times 10^{-7}, \quad \epsilon = 4 \times 10^{-7}, \quad \epsilon = 8 \times 10^{-7} \]
Behavior of the Configurations

\[ P(\text{accept } \Delta E=10) = e^{-10/T(\tau)} \]

\[ \varepsilon = 5 \times 10^{-8} \]
\[ \varepsilon = 1 \times 10^{-7} \]
\[ \varepsilon = 1.5 \times 10^{-7} \]
\[ \varepsilon = 2 \times 10^{-7} \]
\[ \varepsilon = 4 \times 10^{-7} \]
\[ \varepsilon = 8 \times 10^{-7} \]
Behavior of the Configurations

\[ P(\text{accept } \Delta E=50) = e^{-50/T(\tau)} \]
• Our very rough calculations gave us parameter settings for $T_s$ and $\varepsilon$ that produce these temperature- and probability curves.
Our very rough calculations gave us parameter settings for $T_s$ and $\varepsilon$ that produce these temperature- and probability curves.

Whether these settings are actually any good must be studied now.
Relation of $\varepsilon$ and Performance

best f / lb$^*$

sa_exp_20_\varepsilon_1swap

- abz7 / 656
- la24 / 935
- swv15 / 2885
- yn4 / 929

$\varepsilon * 10^7$
Relation of $\varepsilon$ and Performance

- Indeed, values of $\varepsilon \in [1 \times 10^{-7}, 2 \times 10^{-7}]$ perform well for $T_s = 20$. 
Relation of $\varepsilon$ and Performance

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

Only for la24, smaller $\varepsilon$ 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

<table>
<thead>
<tr>
<th>I</th>
<th>algo</th>
<th>makespan</th>
<th>last improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>best</td>
<td>mean</td>
</tr>
<tr>
<td>abz7</td>
<td>hcr_65536_nswap</td>
<td>712</td>
<td>731</td>
</tr>
<tr>
<td></td>
<td>eac_4_5%_nswap</td>
<td>672</td>
<td>690</td>
</tr>
<tr>
<td></td>
<td>sa_exp_20_2_1swap</td>
<td>663</td>
<td>673</td>
</tr>
<tr>
<td>la24</td>
<td>hcr_65536_nswap</td>
<td>942</td>
<td>973</td>
</tr>
<tr>
<td></td>
<td>eac_4_5%_nswap</td>
<td>935</td>
<td>963</td>
</tr>
<tr>
<td></td>
<td>sa_exp_20_2_1swap</td>
<td>938</td>
<td>949</td>
</tr>
<tr>
<td>swv15</td>
<td>hcr_65536_nswap</td>
<td>3740</td>
<td>3818</td>
</tr>
<tr>
<td></td>
<td>eac_4_5%_nswap</td>
<td>3102</td>
<td>3220</td>
</tr>
<tr>
<td></td>
<td>sa_exp_20_2_1swap</td>
<td>2936</td>
<td>2994</td>
</tr>
<tr>
<td>yn4</td>
<td>hcr_65536_nswap</td>
<td>1068</td>
<td>1109</td>
</tr>
<tr>
<td></td>
<td>eac_4_5%_nswap</td>
<td>1000</td>
<td>1038</td>
</tr>
<tr>
<td></td>
<td>sa_exp_20_2_1swap</td>
<td>973</td>
<td>985</td>
</tr>
</tbody>
</table>
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?

\( \text{sa}_\text{exp}_20_2_1\text{swap} \): median result of 3 min of Simulated Annealing with exponential schedule, \( T_s = 20 \), and \( \varepsilon = 2 \cdot 10^{-7} \) and 1\text{swap} 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?

\texttt{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 \texttt{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

swv15 / 3224
So what do we get?

\textbf{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 1\text{swap} 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?

\texttt{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 1\text{swap} unary operator.

\begin{center}
\begin{tabular}{cccccccccccccccccccc}
16 & 0 & 17 & 7 & 19 & 3 & 5 & 2 & 18 & 4 & 1 & 13 & 14 & 12 & 6 & 9 & 8 & 16 & 10 & 18 & 8 & 14 & 4 & 16 & 5 \\
14 & 12 & 3 & 13 & 19 & 4 & 18 & 1 & 17 & 9 & 16 & 2 & 0 & 6 & 15 & 5 & 7 & 8 & 11 & 1 & 17 & 11 & 9 \\
13 & 9 & 10 & 17 & 13 & 15 & 16 & 5 & 2 & 6 & 7 & 8 & 3 & 4 & 1 & 11 & 0 & 7 & 0 & 17 & 17 & 9 & 9 \\
11 & 1 & 7 & 10 & 0 & 12 & 18 & 5 & 8 & 18 & 15 & 19 & 13 & 2 & 17 & 1 & 14 & 16 & 57 & 6 & 14 & 14 \\
10 & 19 & 1 & 7 & 10 & 0 & 12 & 18 & 5 & 8 & 18 & 15 & 19 & 13 & 2 & 17 & 1 & 14 & 16 & 57 & 6 & 14 & 14 \\
8 & 3 & 14 & 9 & 18 & 0 & 12 & 19 & 15 & 6 & 16 & 7 & 8 & 13 & 12 & 2 & 17 & 1 & 4 & 16 & 57 & 6 & 14 \\
7 & 8 & 17 & 0 & 3 & 12 & 14 & 11 & 4 & 16 & 6 & 7 & 2 & 13 & 18 & 1 & 3 & 5 & 0 & 7 & 6 & 11 & 5 \\
6 & 12 & 14 & 8 & 15 & 0 & 9 & 10 & 13 & 16 & 2 & 4 & 18 & 17 & 19 & 3 & 3 & 4 & 5 & 6 & 16 & 11 & 10 \\
5 & 3 & 14 & 18 & 9 & 17 & 2 & 16 & 0 & 15 & 6 & 18 & 917 & 10 & 119 & 12 & 8 & 4 & 7 & 6 & 11 & 5 \\
4 & 13 & 17 & 6 & 8 & 9 & 7 & 5 & 12 & 1 & 11 & 8 & 11 & 7 & 17 & 19 & 0 & 2 & 4 & 10 & 2 & 10 \\
3 & 15 & 9 & 4 & 10 & 7 & 13 & 3 & 17 & 1 & 8 & 5 & 6 & 2 & 18 & 11 & 0 & 7 & 4 & 10 & 2 & 10 \\
1 & 7 & 2 & 5 & 9 & 10 & 16 & 3 & 18 & 2 & 1 & 17 & 14 & 15 & 9 & 10 & 9 & 7 & 2 & 11 & 19 & 17 & 9 & 4 & 50 & 7 \\
0 & 7 & 13 & 0 & 10 & 19 & 8 & 6 & 3 & 18 & 2 & 1 & 17 & 14 & 15 & 9 & 10 & 9 & 7 & 2 & 11 & 19 & 17 & 9 & 4 & 50 & 7
\end{tabular}
\end{center}
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?

- la24
- hcr_65536_nswap
- eac_4_5%_nswap
- sa_exp_20_2_1swap
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?

- yn4
- hcr_65536_nswap
- eac_4_5%_nswap
- sa_exp_20_2_1swap

Time in ms
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 $\epsilon$ 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 $\epsilon$ 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 $\epsilon$ 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 $\epsilon$ 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 $\epsilon$ 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


