Metaheuristic Optimization

7. Simulated Annealing

Thomas Weise · 汤卫思

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

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

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

Outline

1. Introduction
2. Metropolis Algorithms
3. Simulated Annealing
4. Temperature Scheduling
5. Implementation
6. Summary

website

Section Outline

1. Introduction
2. Metropolis Algorithms
3. Simulated Annealing
4. Temperature Scheduling
5. Implementation
6. Summary

Introduction: Annealing

Introduction: Annealing

- Cold working metal causes/increases defects in crystal structure

Introduction: Annealing

- Cold working metal causes/increases defects in crystal structure
- After cold working, annealing [1] is performed

Introduction: Annealing

- Cold working metal causes/increases defects in crystal structure
- After cold working, annealing [1] is performed
- The metal is heated to, like 0.4 * melting temperature

Introduction: Annealing

- Cold working metal causes/increases defects in crystal structure
- After cold working, annealing [1] is performed
- The metal is heated to, like 0.4 * melting temperature
- Ions inside metal can move around

Introduction: Annealing

- Cold working metal causes/increases defects in crystal structure
- After cold working, annealing [1] is performed
- The metal is heated to, like 0.4 * melting temperature
- Ions inside metal can move around
- Metal is slowly cooled down, ions assume low-energy, stable positions in crystal $\rightarrow$ metal becomes more stable

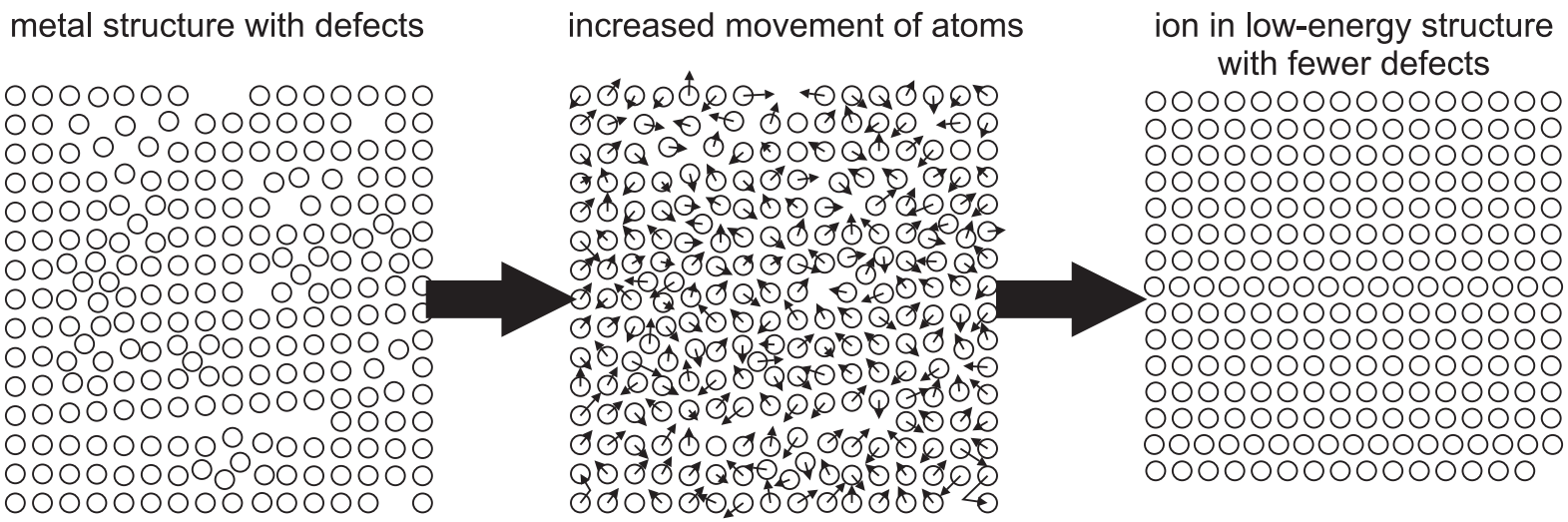
Introduction: Annealing

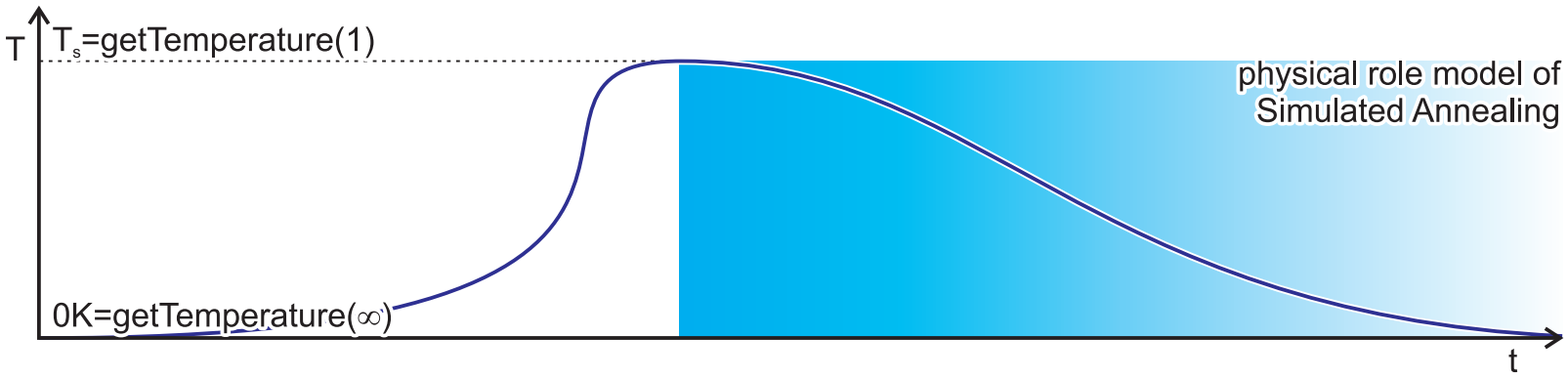
- Cold working metal causes/increases defects in crystal structure
- After cold working, annealing [1] is performed
- The metal is heated to, like 0.4 * melting temperature
- Ions inside metal can move around
- Metal is slowly cooled down, ions assume low-energy, stable positions in crystal $\rightarrow$ metal becomes more stable
- Due to their movement, ions may temporarily assume positions of high energy

Introduction: Annealing

- Cold working metal causes/increases defects in crystal structure
- After cold working, annealing [1] is performed
- The metal is heated to, like 0.4 * melting temperature
- Ions inside metal can move around
- Metal is slowly cooled down, ions assume low-energy, stable positions in crystal $\rightarrow$ metal becomes more stable
- Due to their movement, ions may temporarily assume positions of high energy
- An initial, brittle crystal structure is transformed to a much better configuration by stepping over good and bad states

Introduction: Annealing





Section Outline

1. Introduction
2. Metropolis Algorithms
3. Simulated Annealing
4. Temperature Scheduling
5. Implementation
6. Summary

Metropolis Algorithm

- Metropolis [2] wants to simulate this process.

Metropolis Algorithm

- Metropolis [2] wants to simulate this process.
- First, we need to understand: What is temperature T?

Metropolis Algorithm

- Metropolis [2] wants to simulate this process.
- First, we need to understand: What is temperature T?
- Each material consists of many different particles (atoms, ions, molecules, etc.)

Metropolis Algorithm

- Metropolis [2] wants to simulate this process.
- First, we need to understand: What is temperature T?
- Each material consists of many different particles
- The micro-state of a material is the tuple of the positions and velocities of all particles – this is uninteresting

Metropolis Algorithm

- Metropolis [2] wants to simulate this process.
- First, we need to understand: What is temperature T?
- Each material consists of many different particles
- The micro-state of a material is the tuple of the positions and velocities of all particles – this is uninteresting
- With each such state, there is an energy E associated

Metropolis Algorithm

- Metropolis [2] wants to simulate this process.
- First, we need to understand: What is temperature T?
- Each material consists of many different particles
- The micro-state of a material is the tuple of the positions and velocities of all particles – this is uninteresting
- With each such state, there is an energy E associated: If many the particles move around quickly, the energy E is high and if they don't move, the energy is low
- Now we consider a value of E as a macro-state of the system

Metropolis Algorithm

- Metropolis [2] wants to simulate this process.
- First, we need to understand: What is temperature T?
- Each material consists of many different particles
- The micro-state of a material is the tuple of the positions and velocities of all particles – this is uninteresting
- With each such state, there is an energy E associated: If many the particles move around quickly, the energy E is high
- Now we consider a value of E as a macro-state of the system and to each such state, there belong many possible micro-states
- A system at temperature T has the probability $e^{-\frac{E}{k_B * T}}$ to be in a macro state with energy E.

Metropolis Algorithm

- Metropolis [2] wants to simulate this process.
- First, we need to understand: What is temperature T?
- Each material consists of many different particles
- The micro-state of a material is the tuple of the positions and velocities of all particles – this is uninteresting
- With each such state, there is an energy E associated: If many the particles move around quickly, the energy E is high
- Now we consider a value of E as a macro-state of the system and to each such state, there belong many possible micro-states
- A system at temperature T has the probability $e^{-\frac{E}{k_B * T}}$ to be in a macro state with energy E.

$$k_b = 1.380650524 * 10^{-27} J/K \text{ is the Boltzmann constant} \quad (1)$$

Metropolis Algorithm

- Metropolis [2] wants to simulate this process.
- First, we need to understand: What is temperature T?
- Each material consists of many different particles
- The micro-state of a material is the tuple of the positions and velocities of all particles – this is uninteresting
- With each such state, there is an energy E associated: If many the particles move around quickly, the energy E is high
- Now we consider a value of E as a macro-state of the system and to each such state, there belong many possible micro-states
- A system at temperature T has the probability $e^{-\frac{E}{k_B * T}}$ to be in a macro state with energy E.
- In other words: The higher the temperature, the higher the chance to be in a high-energy state

Metropolis Algorithm

- Based on this, Metropolis [2] develops a simulation for annealing in form of a Monte Carlo algorithm

Metropolis Algorithm

- Based on this, Metropolis [2] develops a simulation for annealing in form of a Monte Carlo algorithm
- pos be the current configuration of the ions and pos' a possible new configuration, T be the temperature (decreasing over time)

Metropolis Algorithm

- Based on this, Metropolis [2] develops a simulation for annealing in form of a Monte Carlo algorithm
- pos be the current configuration of the ions and pos' a possible new configuration, T be the temperature (decreasing over time)

$$\Delta E = E\left(pos'\right) - E\left(pos\right) \tag{1}$$

Metropolis Algorithm

- Based on this, Metropolis [2] develops a simulation for annealing in form of a Monte Carlo algorithm
- pos be the current configuration of the ions and pos' a possible new configuration, T be the temperature (decreasing over time)

$$\Delta E = E\left(pos'\right) - E\left(pos\right) \tag{1}$$

- ΔE is the energy difference between the states

Metropolis Algorithm

- Based on this, Metropolis [2] develops a simulation for annealing in form of a Monte Carlo algorithm
- pos be the current configuration of the ions and pos' a possible new configuration, T be the temperature (decreasing over time)

$$\Delta E = E\left(pos'\right) - E\left(pos\right) \tag{1}$$

- ΔE is the energy difference between the states

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

Metropolis Algorithm

- Based on this, Metropolis [2] develops a simulation for annealing in form of a Monte Carlo algorithm
- pos be the current configuration of the ions and pos' a possible new configuration, T be the temperature (decreasing over time)

$$\Delta E = E\left(pos'\right) - E\left(pos\right) \tag{1}$$

- ΔE is the energy difference between the states

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

- $P(\Delta E)$ is the probability that the new state pos' will be accepted

Section Outline

1. Introduction
2. Metropolis Algorithms
3. Simulated Annealing
4. Temperature Scheduling
5. Implementation
6. Summary

Simulated Annealing

- Metropolis' simulation [2] shows how physical systems find states of low energy.

Simulated Annealing

- Metropolis' simulation [2] shows how physical systems find states of low energy.
- The (simulated) physical system escapes local optima by accepting worse solutions from time to time.

Simulated Annealing

- Metropolis' simulation [2] shows how physical systems find states of low energy.
- The (simulated) physical system escapes local optima by accepting worse solutions from time to time.
- This could be a remedy for the premature convergence problem of hill climbing!

Simulated Annealing

- Metropolis' simulation [2] shows how physical systems find states of low energy.
- The (simulated) physical system escapes local optima by accepting worse solutions from time to time.
- This could be a remedy for the premature convergence problem of hill climbing!
- Idea developed by Kirkpatrick et al. [3], Černý [4], Jacobs et al. [5, 6], and Pincus [7] independently:

Simulated Annealing

- Metropolis' simulation [2] shows how physical systems find states of low energy.
- The (simulated) physical system escapes local optima by accepting worse solutions from time to time.
- This could be a remedy for the premature convergence problem of hill climbing!
- Idea developed by Kirkpatrick et al. [3], Černý [4], Jacobs et al. [5, 6], and Pincus [7] independently:
- Simulated Annealing = hill climbing + sometimes accept worse states following Metropolis' method [8–11]

Simulated Annealing

- Metropolis' simulation [2] shows how physical systems find states of low energy.
- The (simulated) physical system escapes local optima by accepting worse solutions from time to time.
- This could be a remedy for the premature convergence problem of hill climbing!
- Idea developed by Kirkpatrick et al. [3], Černý [4], Jacobs et al. [5, 6], and Pincus [7] independently:
- Simulated Annealing = hill climbing + sometimes accept worse states following Metropolis' method [8–11]
- ⇒ lower risk of premature convergence

Simulated Annealing

- Modification of the Metropolis procedure:

$$\Delta E = f(x') - f(x) \tag{3}$$

- ΔE is the objective value difference between the new (x') and old candidate solution (x)

Simulated Annealing

- Modification of the Metropolis procedure:

$$\Delta E = f(x') - f(x) \tag{3}$$

- ΔE is the objective value difference between the new (x') and old candidate solution (x)

$$P(\Delta E) = \begin{cases} e^{-\frac{\Delta E}{k_B * T}} & \text{if } \Delta E > 0 \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

- $P(\Delta E)$ is the probability that the new candidate solution x' will be accepted

Simulated Annealing

- Modification of the Metropolis procedure:

$$\Delta E = f(x') - f(x) \tag{3}$$

- ΔE is the objective value difference between the new (x') and old candidate solution (x)

$$P(\Delta E) = \begin{cases} e^{-\frac{\Delta E}{\cancel{k_B} * T}} & \text{if } \Delta E > 0 \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

- $P(\Delta E)$ is the probability that the new candidate solution x' will be accepted
- k_B is eliminated from the equation since it is useless for optimization and just makes the acceptance probability of solutions with worse objective values hard to understand

Simulated Annealing

- Modification of the Metropolis procedure:

$$\Delta E = f(x') - f(x) \tag{3}$$

- ΔE is the objective value difference between the new (x') and old candidate solution (x)

$$P(\Delta E) = \begin{cases} e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

- $P(\Delta E)$ is the probability that the new candidate solution x' will be accepted
- k_B is eliminated from the equation since it is useless for optimization and just makes the acceptance probability of solutions with worse objective values hard to understand
- Temperature T reduced according to a specific schedule over the iterations

$p_{best} \longleftarrow \text{simulatedAnnealing}(f)$

Input: f: the objective function to be minimized
Data: p_{new}: the newly generated individual
Data: p_{cur}: the point currently investigated
Data: T: the temperature of the system which is decreased over time
Data: t: the current time index
Data: ΔE: the energy (objective value) difference of the $p_{cur}.x$ and $p_{new}.x$
Output: p_{best}: the best individual ever discovered

```
begin
    p_cur ⟵ create and evaluate initial solution
    p_best ⟵ p_cur
    t ⟵ 1
    while ¬shouldTerminate do
        p_new ⟵ derive new solution from p_cur
        ΔE ⟵ f(p_new) − f(p_cur)
        if ΔE ≤ 0 then
            p_cur ⟵ p_new
            if f(p_cur.x) < f(p_best.x) then p_best ⟵ p_cur
        else
            T ⟵ getTemperature(t)
            if {randomly from [0,1]} < e^(−ΔE/T) then p_cur ⟵ p_new
        t ⟵ t + 1
    return p_best
```

- This is the simplified algorithm, see next slide for full algorithm.

$p_{best} \longleftarrow \text{simulatedAnnealing}(f)$

Input: f: the objective function to be minimized
Data: p_{new}: the newly generated individual
Data: p_{cur}: the point currently investigated
Data: T: the temperature of the system which is decreased over time
Data: t: the current time index
Data: ΔE: the energy (objective value) difference of the $p_{cur}.x$ and $p_{new}.x$
Output: p_{best}: the best individual ever discovered

```
begin
    p_cur ⟵ create and evaluate initial solution
    p_best ⟵ p_cur
    t ⟵ 1
    while ¬shouldTerminate do
        p_new ⟵ derive new solution from p_cur
        ΔE ⟵ f(p_new) − f(p_cur)
        if ΔE ≤ 0 then
            p_cur ⟵ p_new
            if f(p_cur.x) < f(p_best.x) then p_best ⟵ p_cur
        else
            T ⟵ getTemperature(t)
            if {randomly from [0,1]} < e^(−ΔE/T) then p_cur ⟵ p_new
        t ⟵ t + 1
    return p_best
```

- This is the simplified algorithm, see next slide for full algorithm.
- Temperature schedule getTemperature: How the temperature T decreases over time?

$p_{best} \longleftarrow \text{simulatedAnnealing}(f)$

Input: f: the objective function to be minimized
Data: p_{new}: the newly generated individual
Data: p_{cur}: the point currently investigated
Data: T: the temperature of the system which is decreased over time
Data: t: the current time index
Data: ΔE: the energy (objective value) difference of the $p_{cur}.x$ and $p_{new}.x$
Output: p_{best}: the best individual ever discovered

```
begin
    p_cur ⟵ create and evaluate initial solution
    p_best ⟵ p_cur
    t ⟵ 1
    while ¬shouldTerminate do
        p_new ⟵ derive new solution from p_cur
        ΔE ⟵ f(p_new) − f(p_cur)
        if ΔE ≤ 0 then
            p_cur ⟵ p_new
            if f(p_cur.x) < f(p_best.x) then p_best ⟵ p_cur
        else
            T ⟵ getTemperature(t)
            if {randomly from [0,1]} < e^(−ΔE/T) then p_cur ⟵ p_new
        t ⟵ t + 1
    return p_best
```

- This is the simplified algorithm, see next slide for full algorithm.
- Temperature schedule getTemperature: How the temperature T decreases over time?
- $P(\Delta E) = \begin{cases} e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \\ 1 & \text{otherwise} \end{cases}$

$p_{best} \longleftarrow \text{simulatedAnnealing}(f)$

Input: f: the objective function to be minimized
Data: p_{new}: the newly generated individual
Data: p_{cur}: the point currently investigated
Data: T: the temperature of the system which is decreased over time
Data: t: the current time index
Data: ΔE: the energy (objective value) difference of the $p_{cur}.x$ and $p_{new}.x$
Output: p_{best}: the best individual ever discovered

```
begin
    p_cur.g ⟵ create()
    p_cur.x ⟵ gpm(p_cur.g)
    p_cur.y ⟵ f(p_cur.x)
    p_best ⟵ p_cur
    t ⟵ 0
    while ¬shouldTerminate do
        p_new.g ⟵ mutation(p_cur.g)
        p_new.x ⟵ gpm(p_new.g)
        p_new.y ⟵ f(p_new.x)
        ΔE ⟵ p_new.y − p_cur.y
        if ΔE ≤ 0 then
            p_cur ⟵ p_new
            if p_cur.y < p_best.y then p_best ⟵ p_cur
        else
            T ⟵ getTemperature(t)
            if {randomly from [0,1]} < e^(−ΔE/T) then p_cur ⟵ p_new
        t ⟵ t + 1
    return p_best
```

- Full algorithm

Section Outline

1. Introduction
2. Metropolis Algorithms
3. Simulated Annealing
4. Temperature Scheduling
5. Implementation
6. Summary

Temperature Scheduling

Definition (Temperature Schedule)

The temperature schedule defines how the temperature parameter T in the Simulated Annealing process is set. The operator $\mathrm{getTemperature} : \mathbb{N}_1 \mapsto \mathbb{R}^+$ maps the current iteration index t to a (positive) real temperature value T. [12, 13]

Definition (Temperature Schedule)

The temperature schedule defines how the temperature parameter T in the Simulated Annealing process is set. The operator $\mathrm{getTemperature} : \mathbb{N}_1 \mapsto \mathbb{R}^+$ maps the current iteration index t to a (positive) real temperature value T. [12, 13]

$$\mathrm{getTemperature}(t) \in (0, +\infty) \qquad \forall t \in \mathbb{N}_1 \tag{7}$$

Temperature Scheduling

Definition (Temperature Schedule)

The temperature schedule defines how the temperature parameter T in the Simulated Annealing process is set. The operator $\mathrm{getTemperature} : \mathbb{N}_1 \mapsto \mathbb{R}^+$ maps the current iteration index t to a (positive) real temperature value T. [12, 13]

$$\mathrm{getTemperature}(t) \in (0, +\infty) \qquad \forall t \in \mathbb{N}_1 \tag{5}$$

$$\mathrm{getTemperature}(0) = T_{\mathrm{start}} > 0 \tag{7}$$

Temperature Scheduling

Definition (Temperature Schedule)

The temperature schedule defines how the temperature parameter T in the Simulated Annealing process is set. The operator $\mathrm{getTemperature} : \mathbb{N}_1 \mapsto \mathbb{R}^+$ maps the current iteration index t to a (positive) real temperature value T. [12, 13]

$$\mathrm{getTemperature}(t) \in (0, +\infty) \qquad \forall t \in \mathbb{N}_1 \tag{5}$$

$$\mathrm{getTemperature}(0) = T_{\mathrm{start}} > 0 \tag{6}$$

$$\lim_{t \to +\infty} \mathrm{getTemperature}(t) = 0 \tag{7}$$

Temperature Scheduling

Definition (Temperature Schedule)

The temperature schedule defines how the temperature parameter T in the Simulated Annealing process is set. The operator $\mathrm{getTemperature} : \mathbb{N}_1 \mapsto \mathbb{R}^+$ maps the current iteration index t to a (positive) real temperature value T. [12, 13]

$$\mathrm{getTemperature}(t) \in (0, +\infty) \qquad \forall t \in \mathbb{N}_1 \tag{5}$$

$$\mathrm{getTemperature}(0) = T_{\mathrm{start}} > 0 \tag{6}$$

$$\lim_{t \to +\infty} \mathrm{getTemperature}(t) = 0 \tag{7}$$

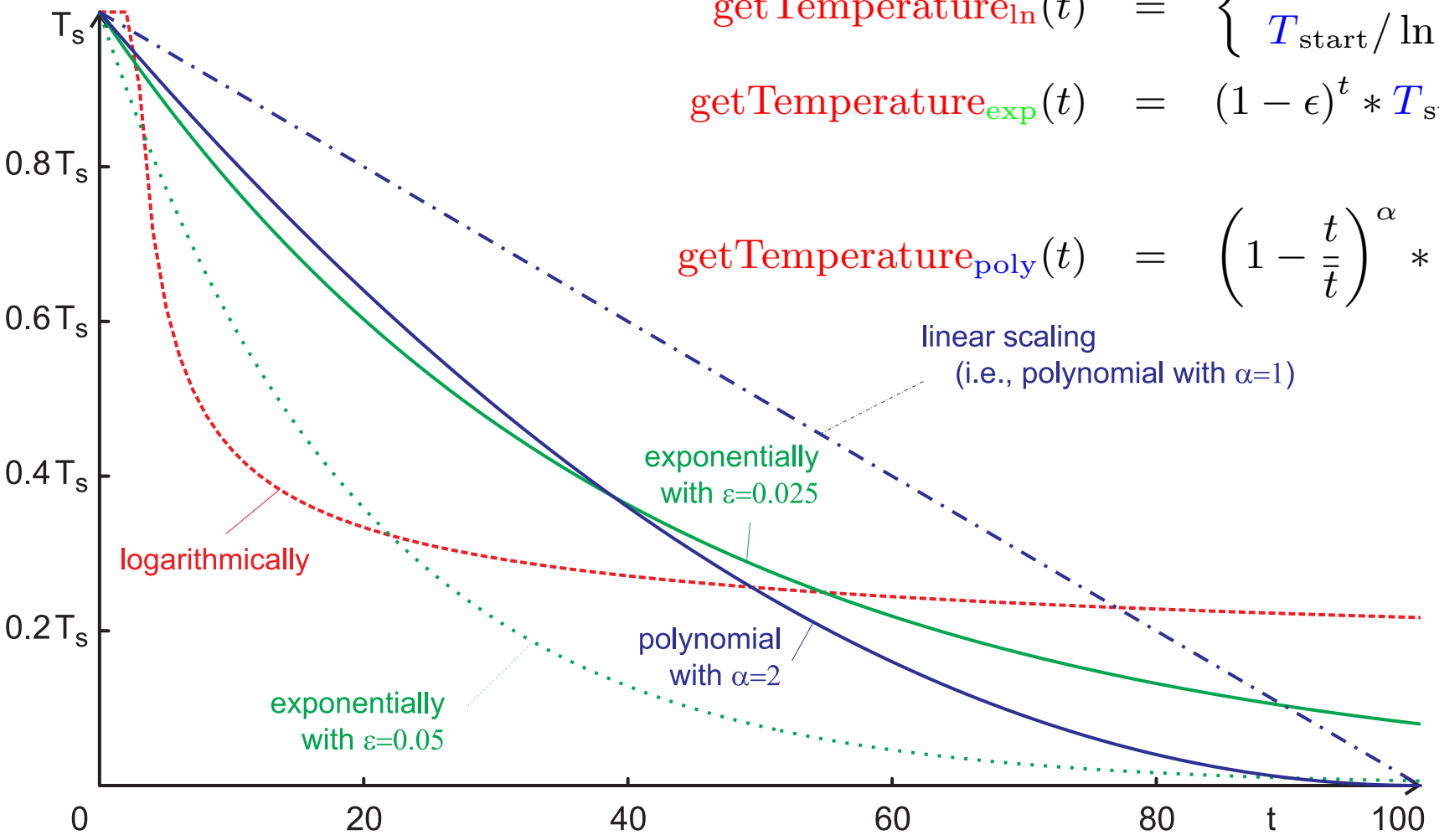
The temperature schedule allows for a smooth transition of SA algorithm behavior from "like Random Walk" (high temperature) to "like hill climbing" (low temperature).

Temperature Scheduling

$$\text{getTemperature}_{\text{ln}}(t) = \begin{cases} T_{\text{start}} & \text{if } t < 3 \\ T_{\text{start}} / \ln t & \text{otherwise} \end{cases}$$

$$\text{getTemperature}_{\text{exp}}(t) = (1-\epsilon)^t * T_{\text{start}}$$

$$\text{getTemperature}_{\text{poly}}(t) = \left(1 - \frac{t}{\bar{t}}\right)^{\alpha} * T_{\text{start}}$$



Temperature Scheduling: Setup

- Logarithmic Scheduling
  - T_{start}: use a value larger than the greatest difference of the objective value of a local minimum and its best neighboring candidate solution

Temperature Scheduling: Setup

- Logarithmic Scheduling
  - T_{start}: use a value larger than the greatest difference of the objective value of a local minimum and its best neighboring candidate solution
- Exponential Scheduling
  - determine $\epsilon \in (0, 1)$ by experiment

Temperature Scheduling: Setup

- Logarithmic Scheduling
  - T_{start}: use a value larger than the greatest difference of the objective value of a local minimum and its best neighboring candidate solution
- Exponential Scheduling
  - determine $\epsilon \in (0, 1)$ by experiment
- Polynomial Scheduling
  - α is a constant, maybe 1, 2, or 4
  - an upper iteration limit $\bar{t}$ after which the temperature should become zero

Temperature Scheduling: Setup

- Logarithmic Scheduling
  - T_{start}: use a value larger than the greatest difference of the objective value of a local minimum and its best neighboring candidate solution
- Exponential Scheduling
  - determine $\epsilon \in (0, 1)$ by experiment
- Polynomial Scheduling
  - α is a constant, maybe 1, 2, or 4
  - an upper iteration limit $\bar{t}$ after which the temperature should become zero
- Adaptive Scheduling
  - Example: $T = \beta * (f(p_{cur}.x) - f(\tilde{x}))$ every m steps, β determined by experiment

Temperature Scheduling: Setup

- Logarithmic Scheduling
  - T_{start}: use a value larger than the greatest difference of the objective value of a local minimum and its best neighboring candidate solution
- Exponential Scheduling
  - determine $\epsilon \in (0, 1)$ by experiment
- Polynomial Scheduling
  - α is a constant, maybe 1, 2, or 4
  - an upper iteration limit $\bar{t}$ after which the temperature should become zero
- Adaptive Scheduling
  - Example: $T = \beta * (f(p_{cur}.x) - f(\tilde{x}))$ every m steps, β determined by experiment
- Often: Adjust temperature only every $m \in \mathbb{N}_1$ steps

Temperature Scheduling: Convergence

- If temperature decreases slowly (e.g., logarithmically), convergence to the global optimum has been proven for various optimization problems

Temperature Scheduling: Convergence

- If temperature decreases slowly (e.g., logarithmically), convergence to the global optimum has been proven for various optimization problems
- ... but the number of function evaluations needed to find the optimum with $P \to 1$ is still higher than what an exhaustive enumeration would need [14]

Temperature Scheduling: Convergence

- If temperature decreases slowly (e.g., logarithmically), convergence to the global optimum has been proven for various optimization problems
- ...but the number of function evaluations needed to find the optimum with $P \to 1$ is still higher than what an exhaustive enumeration would need [14]
- ...which makes sense because otherwise we could solve $\mathcal{NP}$-complete problems efficiently and exactly with SA [15]...

Temperature Scheduling: Convergence

- If temperature decreases slowly (e.g., logarithmically), convergence to the global optimum has been proven for various optimization problems
- ... but the number of function evaluations needed to find the optimum with $P \to 1$ is still higher than what an exhaustive enumeration would need [14]
- ... which makes sense because otherwise we could solve $\mathcal{NP}$-complete problems efficiently and exactly with SA [15]...
- Faster cooling schedules (e.g., exponential ones) lose guaranteed convergence but progress much faster
- Simulated Annealing turns into Simulated Quenching [13]

Temperature Scheduling: Convergence

- If temperature decreases slowly (e.g., logarithmically), convergence to the global optimum has been proven for various optimization problems
- ... but the number of function evaluations needed to find the optimum with $P \to 1$ is still higher than what an exhaustive enumeration would need [14]
- ... which makes sense because otherwise we could solve $\mathcal{NP}$-complete problems efficiently and exactly with SA [15]...
- Faster cooling schedules (e.g., exponential ones) lose guaranteed convergence but progress much faster
- Simulated Annealing turns into Simulated Quenching [13]
- Here: Restarting good in order avoid premature convergence

Section Outline

1. Introduction
2. Metropolis Algorithms
3. Simulated Annealing
4. Temperature Scheduling
5. Implementation
6. Summary

Implementation

- Add a new interface for temperature scheduling

Implementation

- Add a new interface for temperature scheduling
- Implement Simulated Annealing according to the algorithm

Implementation

- Add a new interface for temperature scheduling
- Implement Simulated Annealing according to the algorithm
- Implement some temperature schedules

Implementation

- Add a new interface for temperature scheduling
- Implement Simulated Annealing according to the algorithm
- Implement some temperature schedules
- Test

From the programmer's perspective, we can say:

Listing: Temperature Schedule getTemperature

```
public interface ITemperatureSchedule {

  public abstract double getTemperature(final int t);
}
```

Listing: Simulated Annealing

```
public class SA<G, X> extends OptimizationAlgorithm<G, X> {
  public Individual<G, X> solve(final IObjectiveFunction<X> f) {
    Individual<G, X> pcur, pnew, pbest;
    double deltaE, T;
    int t;

    pcur = new Individual<>();
    pnew = new Individual<>();
    pbest = new Individual<>();

    t = 1;
    pcur.g = this.nullary.create(this.random);
    pcur.x = this.gpm.gpm(pcur.g);
    pcur.v = f.compute(pcur.x);
    pbest.assign(pcur);

    while (!(this.termination.shouldTerminate())) {
      pnew.g = this.unary.mutate(pcur.g, this.random);
      pnew.x = this.gpm.gpm(pnew.g);
      pnew.v = f.compute(pnew.x);
      deltaE = (pnew.v - pcur.v);

      if (deltaE <= 0d) {
        pcur.assign(pnew);
        if (pnew.v < pbest.v) {
          pbest.assign(pnew);
        }
      } else {
        T = this.temperature.getTemperature(t);
        if (this.random.nextDouble() < Math.exp(-deltaE / T)) {
          pcur.assign(pnew);
        }
      }
      t++;
    }
    return pbest;
  }
}
```

The Logarithmic Temperature Schedule

Listing: The Logarithmic Temperature Schedule

```
public class Logarithmic implements ITemperatureSchedule {
  public double getTemperature(final int t) {
    if (t < 3) {
      return this.Ts;
    }
    return (this.Ts / Math.log(t));
  }
}
```

The Exponential Temperature Schedule

Listing: The Exponential Temperature Schedule

```
public class Exponential implements ITemperatureSchedule {
  public double getTemperature(final int t) {
    return (this.Ts * Math.pow((1d - this.epsilon), t));
  }
}
```

The Polynomial Temperature Schedule

Listing: The Polynomial Temperature Schedule

```
public class Polynomial implements ITemperatureSchedule {
  public double getTemperature(final int t) {
    return (this.Ts * Math.pow(Math.max(0d, (1d - (t / this.tmax))),//
        this.alpha));
  }
}
```

Section Outline

1. Introduction
2. Metropolis Algorithms
3. Simulated Annealing
4. Temperature Scheduling
5. Implementation
6. Summary

Summary

- Annealing is a physical process in metallurgy where low-energy configurations are found

Summary

- Annealing is a physical process in metallurgy where low-energy configurations are found
- This process is simulated with the Metropolis algorithm

Summary

- Annealing is a physical process in metallurgy where low-energy configurations are found
- This process is simulated with the Metropolis algorithm
- which serves as role model for an optimization algorithm: simulated annealing

Summary

- Annealing is a physical process in metallurgy where low-energy configurations are found
- This process is simulated with the Metropolis algorithm
- which serves as role model for an optimization algorithm: simulated annealing
- Different temperature schedules

Summary

- Annealing is a physical process in metallurgy where low-energy configurations are found
- This process is simulated with the Metropolis algorithm
- which serves as role model for an optimization algorithm: simulated annealing
- Different temperature schedules
- Convergence to global optimum is guaranteed for many problems and logarithmic schedules . . . but very slow

Summary

- Annealing is a physical process in metallurgy where low-energy configurations are found
- This process is simulated with the Metropolis algorithm
- which serves as role model for an optimization algorithm: simulated annealing
- Different temperature schedules
- Convergence to global optimum is guaranteed for many problems and logarithmic schedules ... but very slow
- Simulated quenching is faster but loses guaranteed optimality

谢谢

Thank you

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

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

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

Bibliography

Bibliography I

1. F. J. Humphreys and M. Hatherly. *Recrystallization and Related Annealing Phenomena*. Pergamon Materials Series. Amsterdam, The Netherlands: Elsevier Science Publishers B.V., 2004. ISBN 0080441645 and 9780080441641. URL http://books.google.de/books?id=52Gloa7HxGsC.
2. Nicholas Metropolis, Arianna W. Rosenbluth, Marshall Nicholas Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953. doi: 10.1063/1.1699114. URL http://sc.fsu.edu/~beerli/mcmc/metropolis-et-al-1953.pdf.
3. Scott Kirkpatrick, Charles Daniel Gelatt, Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science Magazine*, 220(4598):671–680, May 13, 1983. doi: 10.1126/science.220.4598.671. URL http://fezzik.ucd.ie/msc/cscs/ga/kirkpatrick83optimization.pdf.
4. Vladimír Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, January 1985. doi: 10.1007/BF00940812. URL http://mkweb.bcgsc.ca/papers/cerny-travelingsalesman.pdf. Communicated by S. E. Dreyfus. Also: Technical Report, Comenius University, Mlynská Dolina, Bratislava, Czechoslovakia, 1982.
5. Dean Jacobs, Jan Prins, Peter Siegel, and Kenneth Wilson. Monte carlo techniques in code optimization. *ACM SIGMICRO Newsletter*, 13(4):143–148, December 1982.
6. Dean Jacobs, Jan Prins, Peter Siegel, and Kenneth Wilson. Monte carlo techniques in code optimization. In *International Symposium on Microarchitecture – Proceedings of the 15th Annual Workshop on Microprogramming (MICRO 15)*, pages 143–146, Palo Alto, CA, USA, October 5–7, 1982. Piscataway, NJ, USA: IEEE (Institute of Electrical and Electronics Engineers).
7. Martin Pincus. A monte carlo method for the approximate solution of certain types of constrained optimization problems. *Operations Research (Oper. Res.)*, 18(6):1225–1228, November–December 1970.
8. Peter Salamon, Paolo Sibani, and Richard Frost. *Facts, Conjectures, and Improvements for Simulated Annealing*, volume 7 of *SIAM Monographs on Mathematical Modeling and Computation*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics (SIAM), 2002. ISBN 0898715083 and 9780898715088. URL http://books.google.de/books?id=jhAldlYvClcC.
9. Peter J. M. van Laarhoven and Emile H. L. Aarts, editors. *Simulated Annealing: Theory and Applications*, volume 37 of *Mathematics and its Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1987. ISBN 90-277-2513-6, 978-90-277-2513-4, and 978-90-481-8438-5. URL http://books.google.de/books?id=-IgUab6Dp_IC.

Bibliography II

10. Lawrence Davis, editor. *Genetic Algorithms and Simulated Annealing*. Research Notes in Artificial Intelligence. London, UK: Pitman, 1987. ISBN 0273087711, 0934613443, 9780273087717, and 978-0934613446. URL http://books.google.de/books?id=edfSSAAACAAJ.
11. James C. Spall. *Introduction to Stochastic Search and Optimization*. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester, West Sussex, UK: Wiley Interscience, first edition, June 2003. ISBN 0-471-33052-3, 0-471-72213-8, 978-0-471-33052-3, and 978-0-471-72213-7. URL http://books.google.de/books?id=f66OIvvkKnAC.
12. William T. Vettering, Saul A. Teukolsky, William H. Press, and Brian P. Flannery. *Numerical Recipes in C++ – Example Book – The Art of Scientific Computing*. Cambridge, UK: Cambridge University Press, second edition, February 7, 2002. ISBN 0521750342 and 978-0521750349. URL http://books.google.de/books?id=gwijz-OyIYEC.
13. Lester Ingber. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11):29–57, November 1993. doi: 10.1016/0895-7177(93)90204-C. URL http://www.ingber.com/asa93_sapvt.pdf.
14. Andreas Nolte and Rainer Schrader. A note on the finite time behaviour of simulated annealing. *Mathematics of Operations Research (MOR)*, 25(3):476–484, August 2000. doi: 10.1287/moor.25.3.476.12211. URL http://www.zaik.de/~paper/unzip.html?file=zaik1999-347.ps. Revised version from March 1999.
15. Edgar Anderson. The irises of the gaspé peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935.