Metaheuristic Optimization

5. Hill Climbing

Thomas Weise · 汤卫思

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

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

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

Outline

1. Hill Climbing
2. Problems: Local Optima

website

Metaheuristics

- In Lesson 1: *Introduction*, we have learned the idea of metaheuristics

- In Lesson 1: *Introduction*, we have learned the idea of metaheuristics
  1. Start with one (or multiple) initially generated candidate solutions

- In Lesson 1: *Introduction*, we have learned the idea of metaheuristics
  1. Start with one (or multiple) initially generated candidate solutions
  2. Iteratively refine this/these solution(s) solutions in a loop (change, combine, etc.)

- In Lesson 1: *Introduction*, we have learned the idea of metaheuristics
  1. Start with one (or multiple) initially generated candidate solutions
  2. Iteratively refine this/these solution(s) solutions in a loop (change, combine, etc.)
- In Random Sampling, we actually don't do that, we do not *refine* solutions.

- In Lesson 1: *Introduction*, we have learned the idea of metaheuristics
  1. Start with one (or multiple) initially generated candidate solutions
  2. Iteratively refine this/these solution(s) solutions in a loop (change, combine, etc.)
- In Random Sampling, we actually don't do that, we do not *refine* solutions.
- What could be the easiest possible way to realize the metaheuristic idea?

A local search algorithm solves an optimization problem by iteratively moving from *one* candidate solution to a neighboring candidate solution until a termination criterion is met. [1–4]

"*neighboring*" here means: can be reached by applying a search operation once.

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

Input: f: the objective function subject to minization
Input: [implicit] shouldTerminate: the termination criterion
Data: p_{new}: the new solution to be tested
Output: p_{best}: the best individual ever discovered

begin
- $p_{best}.g \longleftarrow \text{create}()$
- $p_{best}.x \longleftarrow \text{gpm}(p_{best}.g)$
- $p_{best}.y \longleftarrow f(p_{best}.x)$
- **while** $\neg \mathit{shouldTerminate}$ **do**
  - $p_{new}.g \longleftarrow \text{mutation}(p_{best}.g)$
  - $p_{new}.x \longleftarrow \text{gpm}(p_{new}.g)$
  - $p_{new}.y \longleftarrow f(p_{new}.x)$
  - **if** $p_{new}.y \leq p_{best}.y$ **then** $p_{best} \longleftarrow p_{new}$
- **return** p_{best}

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

Input: f: the objective function subject to minization
Input: [implicit] shouldTerminate: the termination criterion
Data: p_{new}: the new solution to be tested
Output: p_{best}: the best individual ever discovered

begin
  $p_{best}.g \longleftarrow \text{create}()$
  $p_{best}.x \longleftarrow \text{gpm}(p_{best}.g)$
  $p_{best}.y \longleftarrow f(p_{best}.x)$
  while $\neg \mathit{shouldTerminate}$ **do**
    $p_{new}.g \longleftarrow \text{mutation}(p_{best}.g)$
    $p_{new}.x \longleftarrow \text{gpm}(p_{new}.g)$
    $p_{new}.y \longleftarrow f(p_{new}.x)$
    if $p_{new}.y \leq p_{best}.y$ **then** $p_{best} \longleftarrow p_{new}$
  return p_{best}

1. create initial candidate solution p_{best}

Hill Climbing

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

Input: f: the objective function subject to minization
Input: [implicit] shouldTerminate: the termination criterion
Data: p_{new}: the new solution to be tested
Output: p_{best}: the best individual ever discovered

begin
- $p_{best}.g \longleftarrow \text{create}()$
- $p_{best}.x \longleftarrow \text{gpm}(p_{best}.g)$
- $p_{best}.y \longleftarrow f(p_{best}.x)$
- **while** $\neg shouldTerminate$ **do**
  - $p_{new}.g \longleftarrow \text{mutation}(p_{best}.g)$
  - $p_{new}.x \longleftarrow \text{gpm}(p_{new}.g)$
  - $p_{new}.y \longleftarrow f(p_{new}.x)$
  - **if** $p_{new}.y \leq p_{best}.y$ **then** $p_{best} \longleftarrow p_{new}$
- **return** p_{best}

1. create initial candidate solution p_{best}
2. derive new solution p_{new} from this solution candidate

Hill Climbing

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

Input: f: the objective function subject to minization
Input: [implicit] shouldTerminate: the termination criterion
Data: p_{new}: the new solution to be tested
Output: p_{best}: the best individual ever discovered

begin
  $p_{best}.g \longleftarrow \text{create}()$
  $p_{best}.x \longleftarrow \text{gpm}(p_{best}.g)$
  $p_{best}.y \longleftarrow f(p_{best}.x)$
  while $\neg shouldTerminate$ **do**
    $p_{new}.g \longleftarrow \text{mutation}(p_{best}.g)$
    $p_{new}.x \longleftarrow \text{gpm}(p_{new}.g)$
    $p_{new}.y \longleftarrow f(p_{new}.x)$
    if $p_{new}.y \leq p_{best}.y$ **then** $p_{best} \longleftarrow p_{new}$
  return p_{best}

1. create initial candidate solution p_{best}
2. derive new solution p_{new} from this solution candidate
3. if p_{new} is better than p_{best}, set $p_{best} = p_{new}$

$p_{best} \longleftarrow \mathrm{hillClimbing}(f)$

Input: f: the objective function subject to minization
Input: [implicit] $\mathrm{shouldTerminate}$: the termination criterion
Data: p_{new}: the new solution to be tested
Output: p_{best}: the best individual ever discovered

begin
  $p_{best}.g \longleftarrow \mathrm{create}()$
  $p_{best}.x \longleftarrow \mathrm{gpm}(p_{best}.g)$
  $p_{best}.y \longleftarrow f(p_{best}.x)$
  while $\neg shouldTerminate$ **do**
    $p_{new}.g \longleftarrow \mathrm{mutation}(p_{best}.g)$
    $p_{new}.x \longleftarrow \mathrm{gpm}(p_{new}.g)$
    $p_{new}.y \longleftarrow f(p_{new}.x)$
    if $p_{new}.y \leq p_{best}.y$ **then** $p_{best} \longleftarrow p_{new}$
  return p_{best}

1. create initial candidate solution p_{best}
2. derive new solution p_{new} from this solution candidate
3. if p_{new} is better than p_{best}, set $p_{best} = p_{new}$
4. go back to 2, until termination criterion is met

- Let us implement and test a hill climber

Implementing Hill Climbing

- Let us implement and test a hill climber
- Common domain: real-valued, continuous optimization problems

Implementing Hill Climbing

- Let us implement and test a hill climber
- Common domain: real-valued, continuous optimization problems, i.e., tasks defined over $\mathbb{R}^n$)
- What do we need?

Implementing Hill Climbing

- Let us implement and test a hill climber
- Common domain: real-valued, continuous optimization problems, i.e., tasks defined over $\mathbb{R}^n$)
- What do we need?
  1. Implement basic hill climbing algorithm

Implementing Hill Climbing

- Let us implement and test a hill climber
- Common domain: real-valued, continuous optimization problems, i.e., tasks defined over $\mathbb{R}^n$)
- What do we need?
  1. Implement basic hill climbing algorithm
  2. Implement nullary search operation for $\mathbb{R}^n$

Implementing Hill Climbing

- Let us implement and test a hill climber
- Common domain: real-valued, continuous optimization problems, i.e., tasks defined over $\mathbb{R}^n$)
- What do we need?
  1. Implement basic hill climbing algorithm
  2. Implement nullary search operation for $\mathbb{R}^n$
  3. Implement unary search operation for $\mathbb{R}^n$

Implementing Hill Climbing

- Let us implement and test a hill climber
- Common domain: real-valued, continuous optimization problems, i.e., tasks defined over $\mathbb{R}^n$)
- What do we need?
  1. Implement basic hill climbing algorithm
  2. Implement nullary search operation for $\mathbb{R}^n$
  3. Implement unary search operation for $\mathbb{R}^n$
  4. Implement suitable termination criterion

Implementing Hill Climbing

- Let us implement and test a hill climber
- Common domain: real-valued, continuous optimization problems, i.e., tasks defined over $\mathbb{R}^n$)
- What do we need?
  1. Implement basic hill climbing algorithm
  2. Implement nullary search operation for $\mathbb{R}^n$
  3. Implement unary search operation for $\mathbb{R}^n$
  4. Implement suitable termination criterion
  5. Implement objective functions f

Implementing HC

Listing: The Hill Climbing Algorithm

```java
public class HC<G, X> extends OptimizationAlgorithm<G, X> {
  public Individual<G, X> solve(final IObjectiveFunction<X> f) {
    Individual<G, X> pstar, pnew;

    pstar = new Individual<>();
    pnew = new Individual<>();
    pstar.g = this.nullary.create(this.random);
    pstar.x = this.gpm.gpm(pstar.g);
    pstar.v = f.compute(pstar.x);

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

      if (pnew.v <= pstar.v) {
        pstar.assign(pnew);
      }
    }
    return pstar;
  }
}
```

Applying Hill Climbing: $\mathbb{G} = \mathbb{X} = \mathbb{R}^n$

- Now we have the algorithm. . .

- Now we have the algorithm… …but still need search operations and objective function

Applying Hill Climbing: $\mathbb{G} = \mathbb{X} = \mathbb{R}^n$

- Now we have the algorithm... ...but still need search operations and objective function
- *Here*: $\mathbb{G} = \mathbb{X} = \mathbb{R}^n$

Applying Hill Climbing: $\mathbb{G} = \mathbb{X} = \mathbb{R}^n$

- Now we have the algorithm... ...but still need search operations and objective function
- *Here*: $\mathbb{G} = \mathbb{X} = [\mathbf{min}, \mathbf{max}]^{\mathbf{n}}$

- Now we have the algorithm... ...but still need search operations and objective function
- *Here*: $\mathbb{G} = \mathbb{X} = [\mathbf{min}, \mathbf{max}]^{\mathbf{n}}$
- We implement this *once*, but can re-use it for many other problems (over $\mathbb{R}^n$)

Listing: A definition for a Subspace of $\mathbb{R}^n$

```
public class Rn {
  /** the maximum coordinate value */
  public final double max;

  /** the minimum coordinate value */
  public final double min;

  /** the dimension */
  public final int dim;
}
```

Search Space + Operators

Listing: A nullary search operation in $\mathbb{R}^n$

```
public final class RnNullaryUniform extends Rn implements
    INullarySearchOperation<double[]> {


  @Override
  public final double[] create(final Random r) {
    final double[] g = new double[this.dim];

    for (int i = g.length; (--i) >= 0;) {
      g[i] = (this.min + (r.nextDouble() * (this.max -
          this.min)));
    }

    return g;
  }
}
```

Listing: A unary search operation in $\mathbb{R}^n$

```
public final class RnUnaryNormal extends Rn implements
    IUnarySearchOperation<double[]> {


  @Override
  public final double[] mutate(final double[] genotype, final Random r) {
    double d;

    final double[] g = genotype.clone();
    final int i = r.nextInt(g.length);

    do {
      d = (g[i] + (r.nextGaussian() * (this.max - this.min) * 0.01d));
    } while ((d < this.min) || (d > this.max));

    g[i] = d;
    return g;
  }
}
```

Listing: An alternative unary search operation in $\mathbb{R}^n$

```
public class RnUnaryNormal2 extends Rn implements IUnarySearchOperation<double[]> {

  @Override
  public double[] mutate(final double[] genotype, final Random r) {
    double d;

    final double[] g = genotype.clone();

    for (int i = g.length; (--i) >= 0;) {
      do {
        d = (g[i] + (r.nextGaussian() * (this.max - this.min) * 0.01d));
      } while ((d < this.min) || (d > this.max));
      g[i] = d;
    }

    return g;
  }
}
```

Applying Hill Climbing: $\mathbb{G} = \mathbb{X} = \mathbb{R}^n$

- Now we have the algorithm... ...but still need search operations and objective function
- *Here*: $\mathbb{G} = \mathbb{X} = [min, max]^n$
- We implement this *once*, but can re-use it for many other problems (over $\mathbb{R}^n$)

Applying Hill Climbing: $\mathbb{G} = \mathbb{X} = \mathbb{R}^n$

- Now we have the algorithm… …but still need search operations and objective function
- *Here*: $\mathbb{G} = \mathbb{X} = [min, max]^n$
- We implement this *once*, but can re-use it for many other problems (over $\mathbb{R}^n$)
- Now let's implement some objective functions…

Applying Hill Climbing: $\mathbb{G} = \mathbb{X} = \mathbb{R}^n$

- Now we have the algorithm. . . . . . but still need search operations and objective function
- *Here*: $\mathbb{G} = \mathbb{X} = [min, max]^n$
- We implement this *once*, but can re-use it for many other problems (over $\mathbb{R}^n$)
- Now let's implement some objective functions. . . common benchmark problems for $\mathbb{R}^n$ are:

- Now we have the algorithm... ...but still need search operations and objective function
- *Here*: $\mathbb{G} = \mathbb{X} = [min, max]^n$
- We implement this *once*, but can re-use it for many other problems (over $\mathbb{R}^n$)
- Now let's implement some objective functions... common benchmark problems for $\mathbb{R}^n$ are:

1. $f(x) = \sum_{i=0}^{n-1} x_i^2$ Sphere Function [5–10]
2. $f(x) = \sum_{i=0}^{n-1} \left(\sum_{j=0}^{i} x_j \right)^2$ Schwefel's Problem 1.2 [5, 9]
3. $f(x) = \max\{|x_i| \; \forall i \in 0..(n-1)\}$ Schwefel's Problem 2.21 [5, 9]
4. $f(x) = -\sum_{i=0}^{n-1} \left(x_i \sin \sqrt{|x_i|} \right)$ Schwefel's Problem 2.26 [5, 9]

Listing: The Sphere Function $f(x) = \sum_{i=1}^{n} x_i^2$

```
/** the sphere function */
public class Sphere extends Rn implements IObjectiveFunction<double[]> {
  @Override
  public final double compute(final double[] x) {
    double s = 0d;

    for (int i = this.dim; (--i) >= 0;) {
      s += (x[i] * x[i]);
    }

    return s;
  }
}
```

Listing: Schwefel's Function 2.21 $f(x) = \max\{|x_i|\ \forall i \in 0..(n-1)\}$

```
/** Schwefel problem 2.21 */
public class Schwefel_2_21 extends Rn implements IObjectiveFunction<double[]> {
  @Override
  public final double compute(final double[] x) {
    double m = 0d;

    for (int i = this.dim; (--i) >= 0;) {
      final double d = Math.abs(x[i]);
      if (d > m) {
        m = d;
      }
    }

    return m;
  }
}
```

Applying Hill Climbing: $\mathbb{G} = \mathbb{X} = \mathbb{R}^n$

- Now we have the algorithm... ...but still need search operations and objective function
- *Here*: $\mathbb{G} = \mathbb{X} = [min, max]^n$
- We implement this *once*, but can re-use it for many other problems (over $\mathbb{R}^n$)
- We have implemented some objective functions...

- Now we have the algorithm... ...but still need search operations and objective function
- *Here*: $\mathbb{G} = \mathbb{X} = [min, max]^n$
- We implement this *once*, but can re-use it for many other problems (over $\mathbb{R}^n$)
- We have implemented some objective functions...
- Now we can test our algorithm!

Running

Listing: A simple test program for HC on a 5d Sphere function

```
/** a simple test class applying the hill climber to a function */
public class HCOnSphere {
  public static void main(final String[] args) {
    final HC<double[], double[]> algorithm;
    final Rn searchSpace;
    Individual<double[], double[]> result;

    algorithm = new HC<>();
    searchSpace = new Rn(-10, 10, 5);

    algorithm.nullary = new RnNullaryUniform(searchSpace);
    algorithm.unary = new RnUnaryNormal(searchSpace);

    for (int i = 1; i < 100; i++) {
      algorithm.termination = new MaxSteps(1000000);
      result = algorithm.solve(new Sphere(searchSpace));

      System.out.println("run␣" + i + "␣has␣result␣quality␣" + result.v);
    }
  }
}
```

image source: [11]

Robocode [11]

image source: [11]

Robocode [11]:

- Programming game in Java

image source: [11]

Robocode [11]:

- Programming game in Java
- Program the "AI" driving a small battle robot

Applying Hill Climbing: Robocode!

image source: [11]

Robocode [11]:

- Programming game in `Java`
- Program the "AI" driving a small battle robot
- Goal: Our robot should survive

image source: [11]

Robocode [11]:

- Programming game in Java
- Program the "AI" driving a small battle robot
- Goal: Our robot should survive...and kill many others!
- Let us take a look at that game...

Applying ill Climbing: Robocode!

- Goal: Create a powerful robot via optimization!

Applying ill Climbing: Robocode!

- Goal: Create a powerful robot via optimization!
- Let us look at the code of an example robot: `sample.Tracker`

Applying ill Climbing: Robocode!

Tracker.java 10.10.2012

```
  /**
   * run:  Tracker's main run function
   */
  public void run() {
    // Set colors
    setBodyColor(new Color(128, 128, 50));
    setGunColor(new Color(50, 50, 20));
    setRadarColor(new Color(200, 200, 70));
    setScanColor(Color.white);
    setBulletColor(Color.blue);

    // Prepare gun
    trackName = null; // Initialize to not tracking anyone
    setAdjustGunForRobotTurn(true); // Keep the gun still when we turn
    gunTurnAmt = 10; // Initialize gunTurn to 10

    // Loop forever
    while (true) {
      // turn the Gun (looks for enemy)
      turnGunRight(gunTurnAmt);
      // Keep track of how long we've been looking
      count++;
      // If we've haven't seen our target for 2 turns, look left
      if (count > 2) {
        gunTurnAmt = -10;
      }
      // If we still haven't seen our target for 5 turns, look right
      if (count > 5) {
        gunTurnAmt = 10;
      }
      // If we *still* haven't seen our target after 10 turns, find another
      target
      if (count > 11) {
        trackName = null;
      }
```

Applying ill Climbing: Robocode!

- Goal: Create a powerful robot via optimization!
- Let us look at the code of an example robot: `sample.Tracker`
- It has lots of numerical constants in it...

Applying ill Climbing: Robocode!

```
trackName = null; // Initialize to not tracking anyone
setAdjustGunForRobotTurn(true); // Keep the gun still when we turn
gunTurnAmt = 10; // Initialize gunTurn to 10

// Loop forever
while (true) {
  // turn the Gun (looks for enemy)
  turnGunRight(gunTurnAmt);
  // Keep track of how long we've been looking
  count++;
  // If we've haven't seen our target for 2 turns, look left
  if (count > 2) {
    gunTurnAmt = -10;
  }
  // If we still haven't seen our target for 5 turns, look right
  if (count > 5) {
    gunTurnAmt = 10;
  }
  // If we *still* haven't seen our target after 10 turns, find another
  target
  if (count > 11) {
    trackName = null;
  }
 }
}
```

Example Robot: Tracker.java

Applying ill Climbing: Robocode!

- Goal: Create a powerful robot via optimization!
- Let us look at the code of an example robot: `sample.Tracker`
- It has lots of numerical constants in it...
- ...who knows whether these values are good?

Applying ill Climbing: Robocode!

- Goal: Create a powerful robot via optimization!
- Let us look at the code of an example robot: `sample.Tracker`
- It has lots of numerical constants in it. . .
- . . . who knows whether these values are good?
- Idea: Turn them into parameters that the hill climber can configure

Applying ill Climbing: Robocode!

OptiBot.java 11.10.2012

```
    while (true) {
      if (this.m_trackName != null) {
        if (this.m_rand.nextDouble() < BEHAVIOR[RANDOM_FIRE]) {
          this.setFire(map(BEHAVIOR[RANDOM_FIRE_POWER], 3d));
        }
      }

      // turn the Gun (looks for enemy)
      this.setTurnGunRight(this.m_gunTurnAmt);
      // Keep track of how long we've been looking
      this.m_count += map(BEHAVIOR[COUNT_ADDER], 1d);
      // If we've haven't seen our target for 2 turns, look left
      if (this.m_count > map(BEHAVIOR[COUNT_LIMIT], 2d)) {
        this.m_gunTurnAmt = (map(BEHAVIOR[GUN_TURN_AMOUNT_OVER_LIMIT],
        -10d));
      }
      // If we still haven't seen our target for 5 turns, look right
      if (this.m_count > (map(BEHAVIOR[COUNT_LIMIT_2], 5d))) {
        this.m_gunTurnAmt = map(BEHAVIOR[INIT_GUN_TURN_AMOUNT], 10d);
      }
      // If we *still* haven't seen our target after 10 turns, find another
      // target
      if (this.m_count > map(BEHAVIOR[COUNT_LIMIT_RESET_TRACK], 10d)) {
        this.m_trackName = null;
      }

      if (BEHAVIOR[RANDOM_01] > this.m_rand.nextDouble()) {
        if (this.getOthers() > 1) {
          this.m_trackName = null;
        }
      }

      this.execute();
    }
```

- Goal: Create a powerful robot via optimization!
- Let us look at the code of an example robot: `sample.Tracker`
- It has lots of numerical constants in it...
- ...who knows whether these values are good?
- Idea: Turn them into parameters that the hill climber can configure...and maybe add one or two actions
- Find the best values for the parameters of that new AI `optibot.OptiBot`!

- This is a complex optimization problem!

Applying ill Climbing: Robocode!

- This is a complex optimization problem!
- Because of our parameterization, $\mathbb{G} \subset \mathbb{R}^n$

- This is a complex optimization problem!
- Because of our parameterization, $\mathbb{G} \subset \mathbb{R}^n$
- But what is the problem/solution space $\mathbb{X}$? And how to evaluate a candidate solution $x \in \mathbb{X}$?

Applying ill Climbing: Robocode!

- This is a complex optimization problem!
- Because of our parameterization, $\mathbb{G} \subset \mathbb{R}^n$
- But what is the problem/solution space $\mathbb{X}$? And how to evaluate a candidate solution $x \in \mathbb{X}$?
- OK, we can let the Robocode [11] environment run pre-defined battles via the command line

Applying ill Climbing: Robocode!

- This is a complex optimization problem!
- Because of our parameterization, $\mathbb{G} \subset \mathbb{R}^n$
- But what is the problem/solution space $\mathbb{X}$? And how to evaluate a candidate solution $x \in \mathbb{X}$?
- OK, we can let the Robocode [11] environment run pre-defined battles via the command line
- But how to pass the parameter values to our `OptiBot`?

- This is a complex optimization problem!
- Because of our parameterization, $\mathbb{G} \subset \mathbb{R}^n$
- But what is the problem/solution space $\mathbb{X}$? And how to evaluate a candidate solution $x \in \mathbb{X}$?
- OK, we can let the Robocode [11] environment run pre-defined battles via the command line
- But how to pass the parameter values to our `OptiBot`?
- Let `OptiBot` load a configuration file when it starts!

Applying ill Climbing: Robocode!

- This is a complex optimization problem!
- Because of our parameterization, $\mathbb{G} \subset \mathbb{R}^n$
- But what is the problem/solution space $\mathbb{X}$? And how to evaluate a candidate solution $x \in \mathbb{X}$?
- OK, we can let the Robocode [11] environment run pre-defined battles via the command line
- But how to pass the parameter values to our `OptiBot`?
- Let `OptiBot` load a configuration file when it starts!
- Candidate solutions $x =$ string with text representation of the `double` values of the parameters

Running

Listing: A Genotype-Phenotype Mapping translating double[] to String

```java
public class RobocodeGPM implements IGPM<double[], String> {
  /** the internal string builder */
  private final StringBuilder m_sb;

  /** the robocode gpm */
  public RobocodeGPM() {
    super();
    this.m_sb = new StringBuilder();
  }


  public String gpm(final double[] genotype) {
    boolean notfirst = false;
    this.m_sb.setLength(0);
    for (final double d : genotype) {
      if (notfirst) {
        this.m_sb.append('␣');
      } else {
        notfirst = true;
      }
      this.m_sb.append(d);
    }
    return this.m_sb.toString();
  }
}
```

Applying Hill Climbing: Robocode!

- What is the objective function $f : \mathbb{X} \mapsto \mathbb{R}$?

- What is the objective function $f : \mathbb{X} \mapsto \mathbb{R}$?
- The battle score!

- What is the objective function $f : \mathbb{X} \mapsto \mathbb{R}$?
- The battle score!...but how to compute it?

- What is the objective function $f : \mathbb{X} \mapsto \mathbb{R}$?
- The battle score!... but how to compute it?

2. Take candidate solution (parameter string) and store it in configuration file

- What is the objective function $f : \mathbb{X} \mapsto \mathbb{R}$?
- The battle score!...but how to compute it?

2. Take candidate solution (parameter string) and store it in configuration file
3. Run Robocode

Applying Hill Climbing: Robocode!

- What is the objective function $f : \mathbb{X} \mapsto \mathbb{R}$?
- The battle score!...but how to compute it?

2. Take candidate solution (parameter string) and store it in configuration file
3. Run Robocode
4. Extract score from a (temporary) result file

- What is the objective function $f : \mathbb{X} \mapsto \mathbb{R}$?
- The battle score!...but how to compute it?
  1. clean up all temporary files from previous invocations
  2. Take candidate solution (parameter string) and store it in configuration file
  3. Run Robocode
  4. Extract score from a (temporary) result file

Configure `OptiBot` so that it can win against the others!

Configure `OptiBot` so that it can win against the others!

- Search space $\mathbb{G}$: Parameter Values $\mathbb{G} = \mathbb{R}^{20}$

Applying Hill Climbing: Robocode!

Configure `OptiBot` so that it can win against the others!

- Search space $\mathbb{G}$: Parameter Values $\mathbb{G} = \mathbb{R}^{20}$
- Solution space $\mathbb{X}$: Configuration String (to be stored in a file)

Applying Hill Climbing: Robocode!

Configure `OptiBot` so that it can win against the others!

- Search space $\mathbb{G}$: Parameter Values $\mathbb{G} = \mathbb{R}^{20}$
- Solution space $\mathbb{X}$: Configuration String (to be stored in a file)
- Genotype-phenotype mapping $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$: simply translate `double[]` to `String`...

Applying Hill Climbing: Robocode!

Configure `OptiBot` so that it can win against the others!

- Search space $\mathbb{G}$: Parameter Values $\mathbb{G} = \mathbb{R}^{20}$
- Solution space $\mathbb{X}$: Configuration String (to be stored in a file)
- Genotype-phenotype mapping $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$: simply translate `double[]` to `String`...
- Objective function

Configure `OptiBot` so that it can win against the others!

- Search space $\mathbb{G}$: Parameter Values $\mathbb{G} = \mathbb{R}^{20}$
- Solution space $\mathbb{X}$: Configuration String (to be stored in a file)
- Genotype-phenotype mapping $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$: simply translate `double[]` to `String`...
- Objective function
  - let `OptiBot` battle 25 times against the best demo-robots: `sample.Tracker`, `sample.SpinBot`, and `sample.Walls`

Applying Hill Climbing: Robocode!

Configure `OptiBot` so that it can win against the others!

- Search space $\mathbb{G}$: Parameter Values $\mathbb{G} = \mathbb{R}^{20}$
- Solution space $\mathbb{X}$: Configuration String (to be stored in a file)
- Genotype-phenotype mapping $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$: simply translate `double[]` to `String` ...
- Objective function
  - let `OptiBot` battle 25 times against the best demo-robots: `sample.Tracker`, `sample.SpinBot`, and `sample.Walls`
  - via call to Robocode

Applying Hill Climbing: Robocode!

Configure `OptiBot` so that it can win against the others!

- Search space $\mathbb{G}$: Parameter Values $\mathbb{G} = \mathbb{R}^{20}$
- Solution space $\mathbb{X}$: Configuration String (to be stored in a file)
- Genotype-phenotype mapping $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$: simply translate `double[]` to `String`...
- Objective function
  - let `OptiBot` battle 25 times against the best demo-robots: `sample.Tracker`, `sample.SpinBot`, and `sample.Walls`
  - via call to Robocode
  - Round the score, because battles are randomized (and minimize $-score$ by convention)

Configure `OptiBot` so that it can win against the others!

- Search space $\mathbb{G}$: Parameter Values $\mathbb{G} = \mathbb{R}^{20}$
- Solution space $\mathbb{X}$: Configuration String (to be stored in a file)
- Genotype-phenotype mapping $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$: simply translate `double[]` to `String`...
- Objective function
  - let `OptiBot` battle 25 times against the best demo-robots: `sample.Tracker`, `sample.SpinBot`, and `sample.Walls`
  - via call to Robocode
  - Round the score, because battles are randomized (and minimize $-score$ by convention)
- Plug this into our existing hill climbing implementation!

- This is a dynamic optimization problem involving a complex simulation

- This is a dynamic optimization problem involving a complex simulation
- But we can tackled it with the things we have already implemented!

- This is a dynamic optimization problem involving a complex simulation
- But we can tackled it with the things we have already implemented!
- And subsequently, with the *better* stuffs we will learn!

- This is a dynamic optimization problem involving a complex simulation
- But we can tackled it with the things we have already implemented!
- And subsequently, with the *better* stuffs we will learn!
- All code and things are (of course) included in the `sources.zip` at the teaching page

Applying Hill Climbing: Robocode!

- This is a dynamic optimization problem involving a complex simulation
- But we can tackled it with the things we have already implemented!
- And subsequently, with the *better* stuffs we will learn!
- All code and things are (of course) included in the `sources.zip` at the teaching page
  - the hill climbing approach in package `metaheuristicOptimization.examples.robocode`

- This is a dynamic optimization problem involving a complex simulation
- But we can tackled it with the things we have already implemented!
- And subsequently, with the *better* stuffs we will learn!
- All code and things are (of course) included in the `sources.zip` at the teaching page
  - the hill climbing approach in package `metaheuristicOptimization.examples.robocode`
  - the Robocode libraries and files in folder `resources/robocode`

Applying Hill Climbing: Robocode!

- This is a dynamic optimization problem involving a complex simulation
- But we can tackled it with the things we have already implemented!
- And subsequently, with the *better* stuffs we will learn!
- All code and things are (of course) included in the `sources.zip` at the teaching page
  - the hill climbing approach in package `metaheuristicOptimization.examples.robocode`
  - the Robocode libraries and files in folder `resources/robocode` (including `OptiBot` under `resources/robocode/robots/optibot`)

Listing: A simple test program for HC on the Robocode problem.

```
public class HCOnRobocode {
  public static void main(final String[] args) {
    final HC<double[], String> hc = new HC<>();
    final Rn rn = new Rn(-1.5d, 1.5d, 20);

    hc.gpm = new RobocodeGPM();
    hc.nullary = new RnNullaryUniform(rn);
    hc.unary = new RnUnaryNormal2(rn);
    hc.termination = new MaxSteps(5000);

    System.out.println(hc.solve(new RobocodeObjective()));
  }
}
```

Applying Hill Climbing: Robocode!

Applying Hill Climbing: Combinatorial Problems?

- How about hill climbers for combinatorial problems?
- Let us try, say, to solve a bin packing problem [12]

Listing: A nullary search operation Creating Permutations.

```
public class PermutationNullaryUniform implements INullarySearchOperation<int[]> {
  /** the length of the permutations */
  public final int n;
  @Override
  public int[] create(final Random r) {
    int i, j, t;

    final int[] g = new int[this.n];

    for (i = this.n; (--i) >= 0;) {
      g[i] = i;
    }

    for (i = this.n; (--i) >= 0;) {
      j = r.nextInt(this.n); // see[13–15]
      t = g[j];
      g[j] = g[i];
      g[i] = t;
    }
    return g;
  }
}
```

Listing: Modify a Permutation by Swapping Elements.

```java
public class PermutationUnarySwap implements IUnarySearchOperation<int[]> {
  @Override
  public int[] mutate(final int[] p, final Random r) {
    int i, j, t;

    final int[] g = p.clone();
    do {
      i = r.nextInt(g.length);
      do {
        j = r.nextInt(g.length);
      } while (j == i);
      t = g[i];
      g[i] = g[j];
      g[j] = t;
    } while (r.nextBoolean());

    return g; // return new permutation
  }
}
```

Objective Function

Listing: A simple objective function for the Bin Packing problem.

```
public class BinPackingObjective implements IObjectiveFunction<int[]> {

  /** the size of the bins */
  public final int binSize;

  /** the sizes of the objects */
  public final int[] objects;
  @Override
  public double compute(final int[] x) {
    int bins, remainingSize, curSize;

    bins = 0; // assume zero bins
    remainingSize = 0; // then there also is no space left in them

    for (final int i : x) { // iterate over all elements in the permutation
      curSize = this.objects[i]; // get size of current element
      if (curSize > remainingSize) { // if element does not fit in current bin
          anymore
        bins++; // open a new bin
        remainingSize = this.binSize; // remaining space = bin size
      }
      remainingSize -= curSize; // put object in bin: remaining size reduced
    }
    return bins; // return total number of bins required
  }
}
```

Running

Listing: A simple test program for HC on the Bin Packing problem.

```java
public class HCOnBinPacking {
  public static void main(final String[] args) {
    final HC<int[], int[]> hc = new HC<>();
    final BinPackingObjective f = BinPackingObjective.EXAMPLE_PROBLEM;

    hc.nullary = new PermutationNullaryUniform(f.objects.length);
    hc.unary = new PermutationUnarySwap();

    System.out.println("Hill␣Climbing");
    for (int i = 1; i < 100; i++) {
      hc.termination = new MaxSteps(100000);
      final Individual<int[], int[]> res = hc.solve(f);
      System.out.println(res.v);
    }
  }
}
```

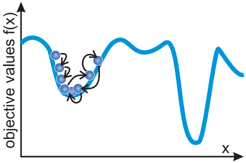
Hill Climbing: Problems

- Hill climbers will "move" through the search space towards better candidate solutions

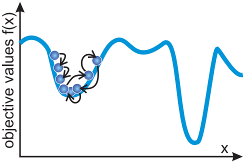
Hill Climbing: Problems

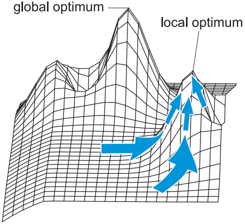
- Hill climbers will "move" through the search space towards better candidate solutions
- What problem could occur if we always and only accept better candidate solutions?

- Hill climbers will "move" through the search space towards better candidate solutions
- What problem could occur if we always and only accept better candidate solutions?



- Hill climbers will "move" through the search space towards better candidate solutions
- What problem could occur if we always and only accept better candidate solutions?





Hill Climbing: Problems

- Hill climbers will “move” through the search space towards better candidate solutions
- What problem could occur if we always and only accept better candidate solutions?

Definition (Premature Convergence)

An optimization process has *prematurely converged* to a local optimum if it is no longer able to explore other parts of the search space than the area currently being examined *and* there exists another region that contains a superior solution [16, 17].

When is optimization effective?

- Now we have: first simple metaheuristic algorithm

When is optimization effective?

- Now we have: first simple metaheuristic algorithm
- Is it good?

When is optimization effective?

- Now we have: first simple metaheuristic algorithm
- Is it good?
- When is optimization efficient?

When is optimization effective?

- Now we have: first simple metaheuristic algorithm
- Is it good?
- When is optimization efficient?
- When the information we get, e.g., from objective function evaluation, is used efficiently

- Now we have: first simple metaheuristic algorithm
- Is it good?
- When is optimization efficient?
- When the information we get, e.g., from objective function evaluation, is used efficiently
- Comparison with algorithms that do not use this information!

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. Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (AIMA)*. Upper Saddle River, NJ, USA: Prentice Hall International Inc. and Upper Saddle River, NJ, USA: Pearson Education, 2nd edition, December 20, 2002. ISBN 0-13-080302-2, 0-13-790395-2, 8120323823, 978-0-13-080302-3, 978-0-13-790395-5, and 9788120323827. URL http://books.google.de/books?id=5mfMAQAACAAJ.
2. Deniz Yuret and Michael de la Maza. Dynamic hill climbing: Overcoming the limitations of optimization techniques. In Selahattin Kuru, editor, *Proceedings of the Second Turkish Symposium on Artificial Intelligence and Neural Networks (Tuärk Yapay Zeka ve Yapay Sinir Ağları Sempozyumu) (TAINN'93)*, pages 208–212, Istanbul, Turkey: Boğaziçi Üniversitesi, June 24–25, 1993. URL http://www.denizyuret.com/pub/tainn93.html.
3. 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.
4. Richard O. Duda, Peter Elliot Hart, and David G. Stork. *Pattern Classification*. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester, West Sussex, UK: Wiley Interscience, 2nd edition, November 2000. ISBN 0-471-05669-3 and 978-0-471-05669-0. URL http://books.google.de/books?id=YoxQAAAAMAAJ.
5. Thomas Weise. *Global Optimization Algorithms – Theory and Application*. Germany: it-weise.de (self-published), 2009. URL http://www.it-weise.de/projects/book.pdf.
6. Ponnuthurai Nagaratnam Suganthan, Nikolaus Hansen, J. J. Liang, Kalyanmoy Deb, Ying-Ping Chen, Anne Auger, and Santosh Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. KanGAL Report 2005005, Kanpur, Uttar Pradesh, India: Kanpur Genetic Algorithms Laboratory (KanGAL), Department of Mechanical Engineering, Indian Institute of Technology Kanpur (IIT), May 2005. URL http://www.iitk.ac.in/kangal/papers/k2005005.pdf.
7. Kenneth Alan De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, Ann Arbor, MI, USA: University of Michigan, August 1975. URL http://cs.gmu.edu/~eclab/kdj_thesis.html.
8. L. Darrell Whitley, Soraya B. Rana, John Dzubera, and Keith E. Mathias. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1-2):245–276, August 1996. doi: 10.1016/0004-3702(95)00124-7. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.134.

9. Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 3(2):82–102, July 1999. doi: 10.1109/4235.771163. URL `http://www.u-aizu.ac.jp/~yliu/publication/tec22r2_online.ps.gz`.
10. Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences – Informatics and Computer Science Intelligent Systems Applications: An International Journal*, 178(15), August 1, 2008. doi: 10.1016/j.ins.2008.02.017. URL `http://nical.ustc.edu.cn/papers/yangtangyao_ins.pdf`.
11. *Robocode: Build the best – destroy the rest!* Fairfax, VA, USA: SourceForge, 2012. URL `http://robocode.sourceforge.net/`.
12. Michael R. Garey and David Stifler Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. New York, NY, USA: W. H. Freeman and Company, 1979. ISBN 0-7167-1044-7, 0-7167-1045-5, 978-0-7167-1044-8, and 978-0-7167-1045-5. URL `http://books.google.de/books?id=mdBxHAAACAAJ`.
13. Sir Ronald Aylmer Fisher and Frank Yates. *Statistical Tables for Biological, Agricultural and Medical Research*. London, UK: Oliver and Boyd and New York, NY, USA: Macmillan Publishers Co., 3rd edition, 1938. ISBN 0-02-844720-4, 0-05-000872-2, 0-582-44525-6, and 978-0582445253. URL `http://digital.library.adelaide.edu.au/dspace/handle/2440/10701`.
14. Richard Durstenfeld. Algorithm 235: Random permutation. *Communications of the ACM (CACM)*, 7(7):420, July 1964. doi: 10.1145/364520.364540.
15. Donald Ervin Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming (TAOCP)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1969. ISBN 0-201-89684-2, 8177583352, 978-0-201-89684-8, and 978-8177583359. URL `http://books.google.de/books?id=OtLNKNVh1XoC`.
16. Rasmus K. Ursem. *Models for Evolutionary Algorithms and Their Applications in System Identification and Control Optimization*. PhD thesis, Århus, Denmark: University of Aarhus, Department of Computer Science, April 1, 2003. URL `http://www.daimi.au.dk/~ursem/publications/RKU_thesis_2003.pdf`.
17. James David Schaffer, Larry J. Eshelman, and Daniel Offutt. Spurious correlations and premature convergence in genetic algorithms. In Bruce M. Spatz and Gregory J. E. Rawlins, editors, *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA'90)*, pages 102–112, Bloomington, IN, USA: Indiana University, Bloomington Campus, July 15–18, 1990. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.