Metaheuristic Optimization

6. Random Walk

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

1. Random Walks

website

Why do Hill Climbers work?

- Hill climbers work under the assumption that there is some structure in the search space.

Why do Hill Climbers work?

- Hill climbers work under the assumption that there is some structure in the search space.
- The chance that a good solution is neighboring another good solution should be higher than that it is surrounded by only bad solutions or at a random location.

Why do Hill Climbers work?

- Hill climbers work under the assumption that there is some structure in the search space.
- The chance that a good solution is neighboring another good solution should be higher than that it is surrounded by only bad solutions or at a random location.
- Based on this idea, the hill climber generates modified copies of the current solution and accepts them if they are better than the old solution.

- Hill climbers work under the assumption that there is some structure in the search space.
- The chance that a good solution is neighboring another good solution should be higher than that it is surrounded by only bad solutions or at a random location.
- Based on this idea, the hill climber generates modified copies of the current solution and accepts them if they are better than the old solution.
- What would happen if we would always accept them?

- Random walks [1–4] are also known as Drunkard's walks

Random Walks

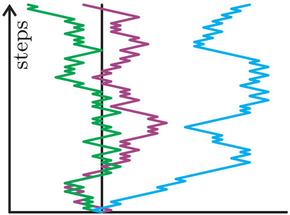
- Random walks [1–4] are also known as Drunkard's walks
- Also do not utilize the information gathered during the search

- Random walks [1–4] are also known as Drunkard's walks
- Also do not utilize the information gathered during the search
- Start at a (random) location

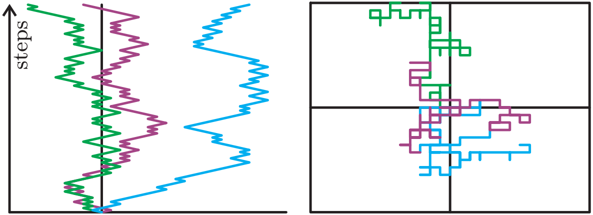
- Random walks [1–4] are also known as Drunkard's walks
- Also do not utilize the information gathered during the search
- Start at a (random) location and take random steps

Random Walks

- Random walks [1–4] are also known as Drunkard's walks
- Also do not utilize the information gathered during the search
- Start at a (random) location and take random steps

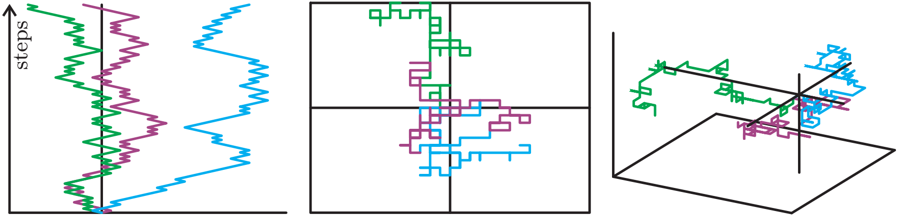


- Random walks [1–4] are also known as Drunkard's walks
- Also do not utilize the information gathered during the search
- Start at a (random) location and take random steps

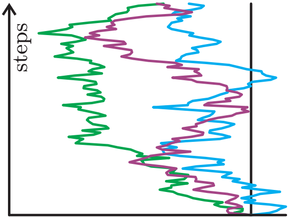


Random Walks

- Random walks [1–4] are also known as Drunkard's walks
- Also do not utilize the information gathered during the search
- Start at a (random) location and take random steps

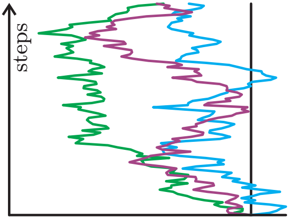


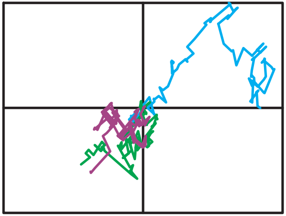
- Random walks [1–4] are also known as Drunkard's walks
- Also do not utilize the information gathered during the search
- Start at a (random) location and take random steps



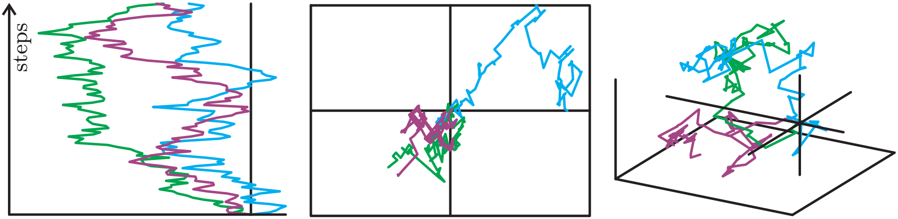
Random Walks

- Random walks [1–4] are also known as Drunkard's walks
- Also do not utilize the information gathered during the search
- Start at a (random) location and take random steps





- Random walks [1–4] are also known as Drunkard's walks
- Also do not utilize the information gathered during the search
- Start at a (random) location and take random steps



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

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

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

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

1. create initial candidate solution p_{best} (also store it in p_{new})

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

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

1. create initial candidate solution p_{best} (also store it in p_{new})
2. derive new solution p_{new} from p_{new}

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

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

1. create initial candidate solution p_{best} (also store it in p_{new})
2. derive new solution p_{new} from p_{new}
3. if p_{new} is better than p_{best}, set $p_{best} = p_{new}$

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

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

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

Random Walks

- Let us implement a random walk

- Let us implement a random walk for
  1. numerical optimization (over $\mathbb{R}^n$) and for

- Let us implement a random walk for
  1. numerical optimization (over $\mathbb{R}^n$) and for
  2. combinatorial optimization (e.g., for TSP over permutations).

Implementing the Random Walk

Listing: The Random Walk Algorithm

```
public class RandomWalk<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);
    pnew.assign(pstar);

    while (!(this.termination.shouldTerminate())) {
      pnew.g = this.unary.mutate(pnew.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;
  }
}
```

- Now we have: first simple metaheuristic algorithm

- 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

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
- Comparison with algorithms that do not use this information!

Summary

- Random walks are like hill climbers, with the exception that they do not use the objective function to guide the search direction.

Summary

- Random walks are like hill climbers, with the exception that they do not use the objective function to guide the search direction.
- Hence, their performance is much worse, similar to random sampling.

Summary

- Random walks are like hill climbers, with the exception that they do not use the objective function to guide the search direction.
- Hence, their performance is much worse, similar to random sampling.
- This means that the idea of expection some sort of "gradient" in the search space towards better solutions is reasonable.

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 I

1. Karl Pearson. The problem of the random walk. *Nature*, 72:294, July 27, 1905. doi: 10.1038/072294b0.
2. William Feller. *An Introduction to Probability Theory and Its Applications, Volume 1*. Wiley Series in Probability and Mathematical Statistics – Applied Probability and Statistics Section Series. Chichester, West Sussex, UK: Wiley Interscience, 3rd edition, 1968. ISBN 0471257087 and 978-0471257080. URL `http://books.google.de/books?id=TkfeSAAACAAJ`.
3. Barry D. Hughes. *Random Walks and Random Environments: Volume 1: Random Walks*. New York, NY, USA: Oxford University Press, Inc., May 16, 1995. ISBN 0-19-853788-3 and 978-0-19-853788-5. URL `http://books.google.de/books?id=QhOen_t0LeQC`.
4. George H. Weiss and Robert J. Rubin. *Random Walks: Theory and Selected Applications*, volume 52 of *Advances in Chemical Physics*. Hoboken, NJ, USA: John Wiley & Sons, Inc., March 14, 2007. ISBN 9780470142769 and 9780471868453. doi: 10.1002/9780470142769.ch5.