合肥学院
HEFEI UNIVERSITY

Metaheuristic Optimization

4. Random Sampling

Thomas Weise · 汤卫思

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

Hefei University, South Campus 2 | 合肥学院 南艳湖校区/南2区
Faculty of Computer Science and Technology | 计算机科学与技术系
Institute of Applied Optimization | 应用优化研究所
230601 Shushan District, Hefei, Anhui, China | 中国 安徽省 合肥市 蜀山区 230601
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99 | 经济技术开发区 锦绣大道99号

Outline

1. Random Sampling

website

- Exhaustive Enumeration has several disadvantages.

- Exhaustive Enumeration has several disadvantages.
- The main problem is that it just needs was too long to be feasible for any practical application.

- Exhaustive Enumeration has several disadvantages.
- The main problem is that it just needs was too long to be feasible for any practical application.
- Another disadvantage is that we need to, well, enumerate the possible solutions, i.e., there must be some kind of order imposed on them.

Random Sampling

- Exhaustive Enumeration has several disadvantages.
- The main problem is that it just needs was too long to be feasible for any practical application.
- Another disadvantage is that we need to, well, enumerate the possible solutions, i.e., there must be some kind of order imposed on them.
- If the solutions are, e.g., real vectors, there is no natural way to do this.

- Exhaustive Enumeration has several disadvantages.
- The main problem is that it just needs was too long to be feasible for any practical application.
- Another disadvantage is that we need to, well, enumerate the possible solutions, i.e., there must be some kind of order imposed on them.
- If the solutions are, e.g., real vectors, there is no natural way to do this.
- Random Sampling is an algorithm which, simply, randomly creates a new candidate solution, usually by uniformly sampling the search space.

Random Sampling

- Exhaustive Enumeration has several disadvantages.
- The main problem is that it just needs was too long to be feasible for any practical application.
- Another disadvantage is that we need to, well, enumerate the possible solutions, i.e., there must be some kind of order imposed on them.
- If the solutions are, e.g., real vectors, there is no natural way to do this.
- Random Sampling is an algorithm which, simply, randomly creates a new candidate solution, usually by uniformly sampling the search space. (*uniform sampling* means that each possible genotype has the same probability of being chosen)

Random Sampling Algorithm

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

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**
    $p_{new}.g \longleftarrow \mathrm{create}()$
    $p_{new}.x \longleftarrow \mathrm{gpm}(p_{new}.g)$
    $p_{new}.y \longleftarrow f(p_{new}.x)$
    if $p_{new}.y \leq p_{best}.y$ **then** $p_{best} \longleftarrow p_{new}$
  return p_{best}

Random Sampling Algorithm

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

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

1. create random candidate solution p_{best}

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

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**
        $p_{new}.g \longleftarrow \mathrm{create}()$
        $p_{new}.x \longleftarrow \mathrm{gpm}(p_{new}.g)$
        $p_{new}.y \longleftarrow f(p_{new}.x)$
        if $p_{new}.y \leq p_{best}.y$ **then** $p_{best} \longleftarrow p_{new}$
    return p_{best}

1. create random candidate solution p_{best}
2. create a *completely new* random solution p_{new}

Random Sampling Algorithm

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

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**
    $p_{new}.g \longleftarrow \mathrm{create}()$
    $p_{new}.x \longleftarrow \mathrm{gpm}(p_{new}.g)$
    $p_{new}.y \longleftarrow f(p_{new}.x)$
    if $p_{new}.y \leq p_{best}.y$ **then** $p_{best} \longleftarrow p_{new}$
  return p_{best}

1. create random candidate solution p_{best}
2. create a *completely new* random solution p_{new}
3. if p_{new} is better than p_{best}, set $p_{best} = p_{new}$

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

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

1. create random candidate solution p_{best}
2. create a *completely new* random solution 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

Implementing the Random Sampling

- Let us implement random sampling

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

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

Implementing the Random Sampling

Listing: The Random Sampling Algorithm

```
public class RandomSampling<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.nullary.create(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;
  }
}
```

Summary

- Random sampling keeps randomly generating new candidate solutions while remembering the best one.

Summary

- Random sampling keeps randomly generating new candidate solutions while remembering the best one.
- If we imagine a large search space with, say, millions of points, then it is clear that it would take very long to (by pure chance) find the (maybe only one) best solution this way.

Summary

- Random sampling keeps randomly generating new candidate solutions while remembering the best one.
- If we imagine a large search space with, say, millions of points, then it is clear that it would take very long to (by pure chance) find the (maybe only one) best solution this way.
- It may even take extremely long to find anything remotely good.

Summary

- Random sampling keeps randomly generating new candidate solutions while remembering the best one.
- If we imagine a large search space with, say, millions of points, then it is clear that it would take very long to (by pure chance) find the (maybe only one) best solution this way.
- It may even take extremely long to find anything remotely good.
- Thus, random sampling is, like exhaustive enumeration, never every use for optimization.

Summary

- Random sampling keeps randomly generating new candidate solutions while remembering the best one.
- If we imagine a large search space with, say, millions of points, then it is clear that it would take very long to (by pure chance) find the (maybe only one) best solution this way.
- It may even take extremely long to find anything remotely good.
- Thus, random sampling is, like exhaustive enumeration, never every use for optimization.
- But this algorithm has two advantages compared to exhaustive enumeration

Summary

- Random sampling keeps randomly generating new candidate solutions while remembering the best one.
- If we imagine a large search space with, say, millions of points, then it is clear that it would take very long to (by pure chance) find the (maybe only one) best solution this way.
- It may even take extremely long to find anything remotely good.
- Thus, random sampling is, like exhaustive enumeration, never every use for optimization.
- But this algorithm has two advantages compared to exhaustive enumeration:
  1. We do not need to impose any order on the search space.

Summary

- Random sampling keeps randomly generating new candidate solutions while remembering the best one.
- If we imagine a large search space with, say, millions of points, then it is clear that it would take very long to (by pure chance) find the (maybe only one) best solution this way.
- It may even take extremely long to find anything remotely good.
- Thus, random sampling is, like exhaustive enumeration, never every use for optimization.
- But this algorithm has two advantages compared to exhaustive enumeration:
  1. We do not need to impose any order on the search space.
  2. Everytime we start it, it will look at the candidate solutions in a different sequence – while exhaustive enumeration, with a poorly chosen order, will always necessarily take very very long.

Summary

- Random sampling keeps randomly generating new candidate solutions while remembering the best one.
- If we imagine a large search space with, say, millions of points, then it is clear that it would take very long to (by pure chance) find the (maybe only one) best solution this way.
- It may even take extremely long to find anything remotely good.
- Thus, random sampling is, like exhaustive enumeration, never every use for optimization.
- But this algorithm has two advantages compared to exhaustive enumeration:
  1. We do not need to impose any order on the search space.
  2. Everytime we start it, it will look at the candidate solutions in a different sequence – while exhaustive enumeration, with a poorly chosen order, will always necessarily take very very long.
- Yet, this algorithm is still entirely useless.

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